YOUR ROBOT

FREE INSIDE EVERY ISSUE

LING COMPUTER PROJECTS MAGAZINE NOVEMBER 1984

ECTRONICS & COMPUTING MONTHLY

AN EMAP PUBLICATION

USA $2.95
Germany D6.00 95p

11 MAC INFO QUERY.

12 DISC DRIVE

21 REF

27

31 REF

WORD PROC

38

55

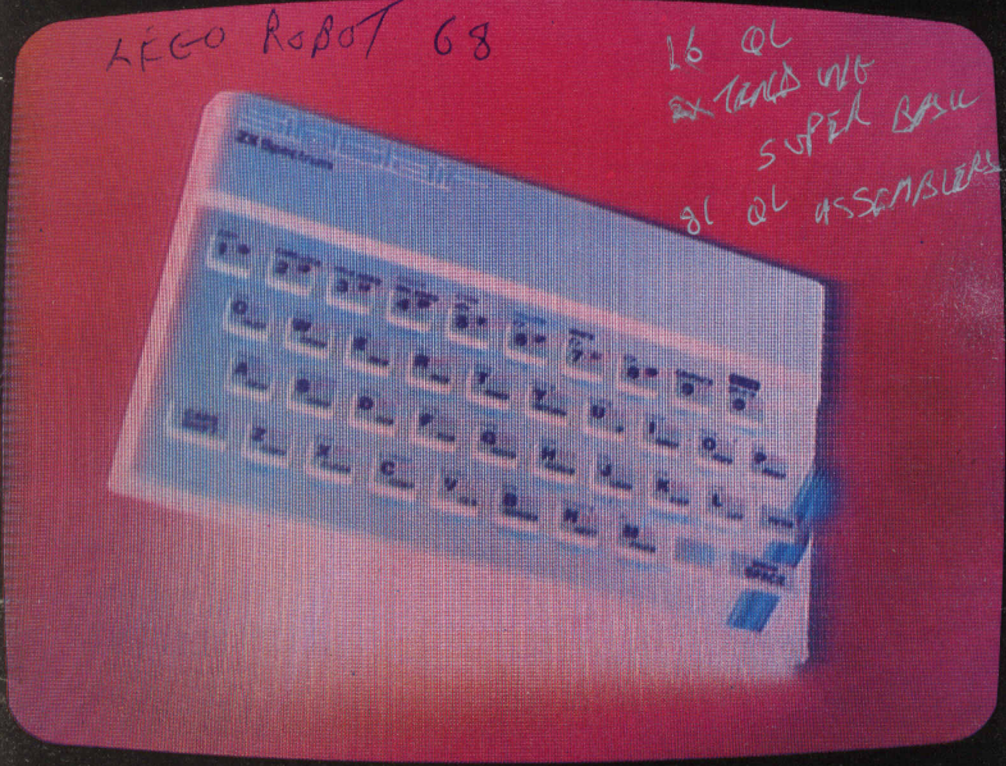
65

67 ROBOT

73 MOTORS

84 BOOKS

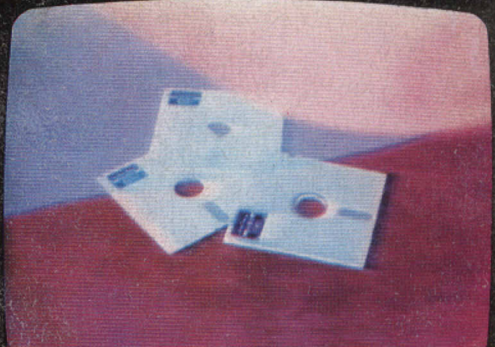
88 LIST

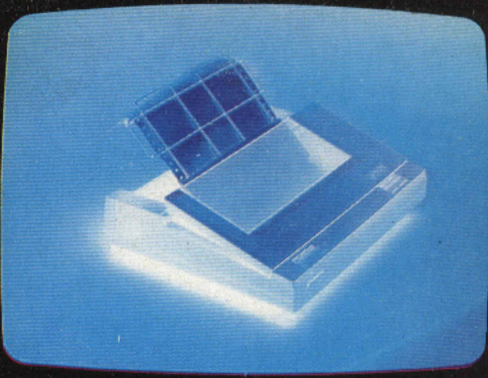


LEGO ROBOT 68

16 QL EXTENDING SUPER BASIC

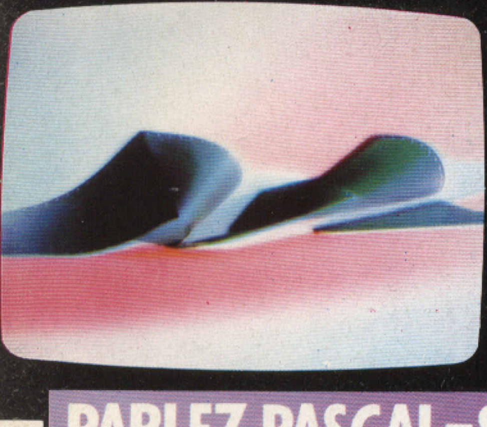
81 QL ASSEMBLERS





WAR OF THE WORDS

WE CHECK OUT THE TOP TEXT PROCESSORS



PARLEZ PASCAL–STRUCTURED PROGRAMMING COURSE

QL M/C UTILITIES • BBC LOGIC STATE ANALYSER

COMMODORE'S PLUS 4 REVIEWED • AMSTRAD I/O PORT

ELECTRONICS & COMPUTING Contents

Vol. 4 Issue 11

COVER PHOTO BY ROB BRIMSON

Electronics & Computing Monthly
Scriptor Court, 155 Farringdon Road,
London, EC1R 3AD

Editorial 01-833-0846
Editor Gary Evans
Deputy Editor William Owen
Production Editor Liz Gregory
Administration Serena Hadley

Advertising 01-833-0531
Advertisement Manager Anthony Herman
Chief Executives Richard Jansz
Philip Baker
Classified Tracey Keighley
Advertising Production Yvonne Moyser

Production 01-833-0531
Art Editor Jeremy Webb
Make-up Time Graphics
Publisher Alfred Rolington

Distribution
EMAP National Publications

Published by
EMAP Business and
Computer Publications

Printed by
Riverside Press, England

Subscriptions
Electronics & Computing Monthly,
(Subscription Department),
Competition House, Farndon Road,
Market Harborough, Leicestershire.

Electronics & Computing Monthly is normally published on the 13th day of each month.

Subscription rates: UK £11.30 incl. post. For overseas rates apply to Subscription Dept., Competition House, Farndon Road, Market Harborough, Leicestershire. Back issues available from: 'EMAP National Publications (*E&CM* Back Numbers), Bretton Court, Peterborough, PE3 8DZ. Phone: 0733 264666.

PROJECTS

FEATURES

REVIEWS

PLUS

And within *Your Robot*
All the latest news in addition to full constructional details of a low cost turtle.

The British Disease

The attitude of British companies towards the requirements of marketing their products leaves a lot of foreign visitors to these shores in a state of some amazement. Many British firms, having spent considerable time and effort in the development of products fail miserably when it comes to creating public awareness of their achievements. Often there seems an in-built belief that the world will beat a path to the door of any company with a value for money/technically innovative product to offer. Sadly, although this is patently not the case, time and time again we see good products failing to achieve their full potential through the lack of any structured approach to marketing.

The computer industry is full of examples that substantiate the point. A notable example is Dragon Data. Previous to their most recent financial troubles and subsequent sale to the Spanish Eurohard organisation, Dragon had a product that was both technically sound and good value for money. The computer was launched into a market that, at the time, had room for a machine of the Dragon 32's specification. In theory, it should have been a low-cost option for people wanting a computer of the BBC micro's quality but who could not afford the £400 price tag on Acorn's machine. Dragon, though, seemed to proceed from banana skin to banana skin with the result that the 32 never had a chance to meet its full promise. It would need a skilled detective to discover any trace of an overall plan behind the marketing operations of Dragon Data.

Not all British firms fail when it comes to telling the world about their product. Sinclair is an obvious exception. Sir Clive, through a skilled and consistent marketing operation created the mass home computer market in this country almost single handed. While some could argue that some aspects of Sinclair's operation were probably the result of luck rather than careful planning, the bottom line is that the end result was a great success. There are a number of UK firms that could benefit from taking a leaf out of Sinclair's book.

The recent PCW show offered many examples of people 'doing it wrong'. A number of household names in the computer industry invested huge sums of money in large stands full of visually interesting displays. In the main, though, the people manning the stands were professional exhibition demonstrators who might just as well have been on a stand at a motor show. Technical questions of even a simple nature were met with a blank expression and the response that the person who could answer was at the bar/back at the office/on holiday or was tied up with someone else. In many cases the money spent on stand space was largely wasted. To an extent, a presence at PCW shows the company is still in business, keeps the dealers happy and must do at least some good. But the point is that, with a little more thought, so much more could have been achieved.

With the home computer market becoming so crowded, and as a result so much more competitive, it is essential that marketing standards improve. Those companies that fail to do so will not be around for too much longer.

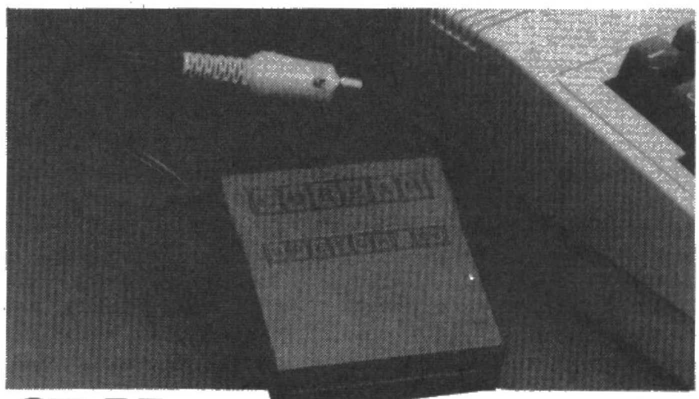
Gary Evans

Video Aided Learning

Master Class have announced the release of two new video cassettes designed to enable VIC 20 owners to get the most from their computers. The level 1 tape illustrates the fundamentals of the computer's operation and BASIC programming while the level 2 cassette moves on to more advanced aspects of the programmer's art including the creation of moving graphics displays.

Both tapes feature audio recordings of programs which may be loaded into the VIC 20 in much the same way as software would be loaded from a standard audio cassette recorder.

Master Class also produce video tapes designed for use with other computers including the Electron, Spectrum, and BBC micro. One title in the BBC series of tapes is aimed at businessmen wishing to explore the potential of the Beeb as a Z80 based business system.



CBM speaks out

The Currah Spectrum μSpeech launched over a year ago broke new ground in the design of low cost speech synthesis systems. The company recently released a similar product, this time designed for operation with the Commodore 64.

The Speech 64 is an allophone based speech unit that manages to overcome the major drawback of similar units by incorporating a unique 'text to speech' interpreter. The majority of allophone speech generators, while offering the potential of unlimited vocabulary, require some fairly difficult programming if they are to produce intelligible output. With the Speech 64 it is only necessary to enter words that are to be sounded in the form of a text string and the interpreter will do the rest.

The unit provides two voice pitches and each of these has two levels of emphasis. In addition to speech output the Speech 64 allows the key presses of the keyboard to be sounded by the synthesiser. This facility is of great benefit to the blind or those with impaired sight.

The unit is easy to install and reproduces the speech output over the loudspeaker of the TV set used as the computer's monitor. The small size of the Speech 64 is a testimony to the art of the ULA, the use of such a device also contributing to the low cost of the product.

The Speech 64 should be in the shops by now and Currah are one company that aim to cash in on the Christmas spending boom.

Excuse my Patent

The fur has certainly been flying around the Cambridge area over the past few months as accusations of piracy and plagiarism move from the realms of computer software to the hardware side of the fence. The trouble centres around the design of the RAM expansion systems for the BBC micro now being marketed by at least three companies. The problem is that one company, Cambridge Computer Consultants (who market the Aries B20), claim to hold a patent which any other product providing additional RAM infringes. To date, Cambridge Computer Consultants have not made copies of their patent application readily available in order that the validity of their assertions over copyright infringement can be examined, although as this issue of *E&CM* goes to press we understand that the patent is soon to be published.

It is certainly difficult to imagine a patent that would be so all-encompassing as to prevent any design for a memory expansion board to fall within its sphere, yet this is apparently how CCC see the situation.

Those people behind the Aries system have reacted to the arrival of other RAM boards in a manner which can do no credit to themselves. In addition to threatening legal action, the company have taken to making claims that the other boards are made with cost cutting techniques and use substandard components. This is something that Texas Instruments, suppliers of some of the ICs used in the non-Aries boards, have taken quite seriously. Indeed the people behind the rival systems can quite legitimately feel that their reputations may be damaged.

In addition, the price of the Aries board to the trade has been reduced to a level that, some estimates show, is at or below cost. Presumably the idea is to kill the opposition off in a short term price war.

Come on Cambridge Computer Consultants, if you do have a patent which other people are infringing, venture out into the open and publish it. Test the validity of your claims in court if necessary. In the meantime it doesn't do your reputation, nor that of the industry's, any good to engage in a mud slinging exercise.

New BBC RAM expansion socket

As an adjunct to the 'Excuse my Patent' story on this page, Raven Micro Products have announced a board that provides 20K of extra RAM for the BBC micro. It allows programs of up to 28K long to be run irrespective of screen mode. The unit is easily installed and does not require any soldering nor cutting of tracks on the micro's PCB.

Only documented MOS calls are used and the board is completely transparent to the BBC computer's operating system.

The unit is supplied with ROM-based software that includes a number of extra commands including a patch to overcome the bug in View 2.1

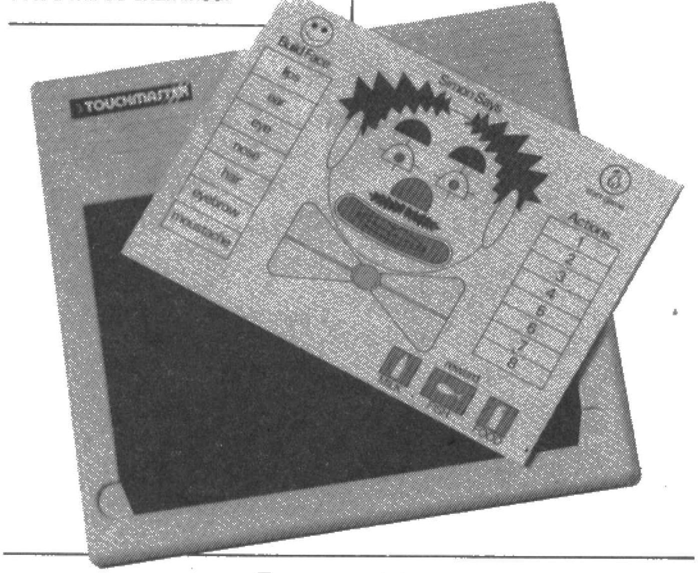
E&CM plans a full review of Raven's product in next month's issue when the full capabilities of the board will be examined.

The Gentle Touch

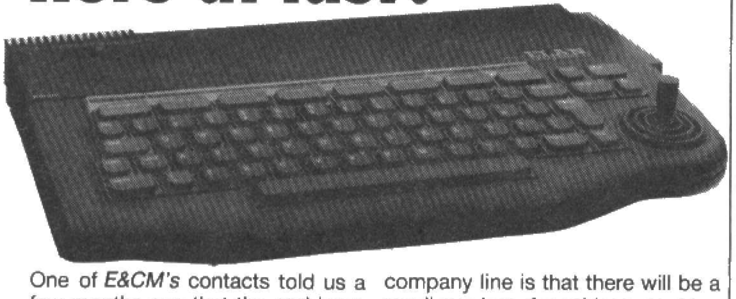
There have been a number of pressure-sensitive graph pads released over the past six months, the latest being the Touchmaster. Touchmaster has an A4 sized working surface offering a resolution of 256 x 256 pixels. The unit incorporates its own microprocessor making the problems of interfacing to a range of computers a straightforward task.

To complement the pad, a range of software going under the name of Touchware has been developed. The first release of Touchware is targeted at the BBC, CBM 64, Dragon, Spectrum, and VIC 20 computers. Initial packages in the range include a graphic package that allows the user to draw, paint and construct designs using preprogrammed shapes.

Touchmaster is to retail at £149.95 and will be available through a range of outlets nationwide.



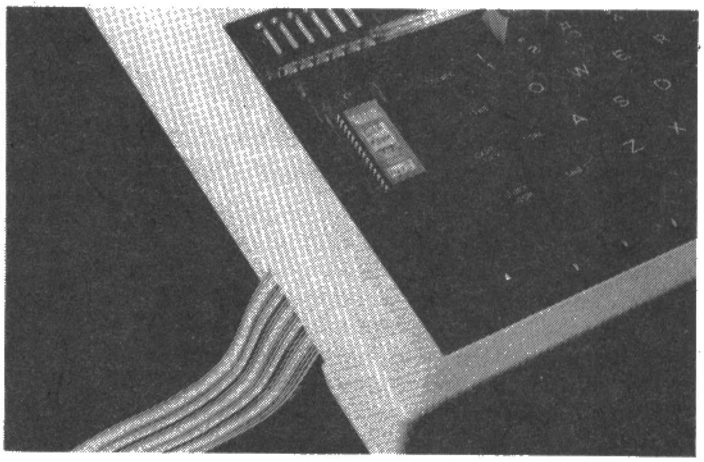
Here at last?



One of *E&CM's* contacts told us a few months ago that the problems that had delayed the Enterprise computer had been overcome and that we could expect to see the machine, at least in small quantities, before Christmas. Unfortunately, to protect the innocent, we were asked not to publish the information.

The prediction turned out to be true and the recent PCW provided an opportunity for the public to have one of their first sightings of the machine that many people thought would never see the light of day. The company line is that there will be a small number of machines reaching the shops over the next few months but full production will not be achieved until 1985. This is not to be the Enterprise's Christmas then and the machine will have to wait until next year before entering the cut-throat mass-market arena.

We have managed to get hold of considerable technical information on the computer and will bring you a full report on the capabilities of the computer in our next issue.



Watford's ZIF ways ROM

Watford Electronics have just released a clever little add-on for the BBC computer. The idea is so simple that it is amazing that no one has thought of it before, but then that is true of so many inventions! The unit is designed to make life easier for those Beeb owners whose collection of sideways ROM software has outgrown the four slot capacity of the computer and perhaps in some cases is starting to put a strain on an expansion board.

The unit consists of a ZIF socket that fits into 'the black hole' to the right of the BBC's keyboard. For those of you who are not familiar with the term, ZIF stands for Zero Insertion Force. The socket is far larger than a standard low profile socket but incorporates a small lever that in one psosition allows ICs to be easily inserted and removed, yet in the other locks them tightly into place. The ribbon cable attached to the socket is terminated in an IC header plug that simply plugs into a vacant sideways ROM slot.

It is thus possible to slide ROMs in and out of the BBC computer with the minimum of fuss. As the IC is visible in the socket it is easy to keep track of exactly which piece of ROM-based software you are working with at any one time.

The ZIF module is less than half the price of some ROM expansion boards and should be an attractive proposition for anyone who has reached the capacity of the BBC's vacant slots but does not wish to meet the cost of an expansion board.

MSX High Street menace

Reports on just how seriously the Japanese are taking the launch of MSX computers in this country vary considerably. Some commentators feel that the oriental consumer electronic giants are merely dabbling in the UK market and are treating this country as a test-bed for more serious moves in the rest of Europe and in the USA. The other school of thought is that the UK is seen as a key market and that there will be an all-out effort to ensure that the MSX standard does find acceptance in Britain.

Supporters of this view will not be surprised at some disturbing, though unconfirmed reports, of some dirty tricks in the High Street. The rumour is that some of the big names behind the MSX flag are suggesting to their dealer networks that, unless they take adequate numbers of MSX computers and give them prominent display positions, then it may prove a little difficult for those same dealers to obtain supplies of the TV/audio equipment on which so many depend upon for their bread and butter income. There is no doubt of the pressure that could be put on the typical outlet and equally no doubt that if, for whatever reason, a significant number of shops start to push MSX in a big way, the computers will stand a far better chance of achieving a significant market share.

We must stress, though, that as yet these are unconfirmed reports. Should we hear anything further though, you'll be the first to know!

IMPORTANT NOTE

We regret that in future we will be unable to answer any technical enquiries over the telephone. All enquiries should be in writing and must be accompanied by a stamped, self - addressed envelope. You should allow at least 14 working days for a reply

Enquiries should be addressed to:

Technical Department
E&CM
Scriptor Court
155 Faringdon Road
London

ACORN VIDEO

Acorn Computers have announced their intention to move into the field of interactive video systems. We have the background to this new industry.

Acorn Computer Group plc have just announced the formation of a new subsidiary company, Acorn Video Ltd, a move which may be as important to the video industry as the Acorn BBC computer contract was to schools and the home computer market.

But what exactly is the video industry? Despite the predominance of street-corner Video Shops it has very little to do with copying films like The Sound of Music and hiring them out for £1.50 per night. What it *has* to do with is the production of educational and training videos for schools, training colleges, commerce and industry, and the production of material for the entertainment industry (pop videos, visuals for arcade games and films on disc). As well as a potentially healthy home market for these products, there is a vast international market – world sales of British films and TV programmes have always helped our balance of payments.

What we are witnessing at the moment is a convergence of technologies, with the power of the micro blending with the impact of the audio-visual image. It is a field so important that equipment manufacturers like Philips, Thorn EMI, RCA and JVC have spent millions of pounds on research and development. And most of this money has gone towards the development of the Video Disc.

Video disc technology has had what you might call an unhappy childhood. Invented in the 1960s and launched to the American public around 1980 and to the British public in 1984, the sales of video disc players haven't yet taken off in a way which would repay the manufacturers' billion dollar investment. Basically, the VHS and Betamax video-cassette recorders (VCRs), occupy an entrenched position in the domestic market place which the read-only Video disc players will have difficulty in breaching.

But this will not always be so. The addition of a microcomputer to control the video disc creates a powerful system, ie Interactive Video, which cannot be realised effectively using VCR players. Acorn Video's Technical Director, Michael Grove, who has pioneered work in Britain on interfacing the standard video disc player to the small computer, states that Acorn Video intends "to direct the interests of the Acorn computer user towards the area of converged technology". Specifically; this means that from October this year you will be able to link your BBC micro to one of six commercial Video disc players, provided you obtain the necessary interface hardware. This consists of a Genlock Board, sideways ROM, software on disc, and connecting leads. You will also need to upgrade the micro by adding twin floppy-disc drives and colour monitor if your present system doesn't include these items.

Next month, we will be taking an in-depth look at the technology involved in Interactive Discs. ■

E&CM PCB SERVICE

September 1983	
BBC Darkroom Timer	**£1.50**
October 1983	
Spectrum Effects Box	**£2.22**
Cassette Signal Conditioner	**£1.60**
BBC EPROM Programmer	**£6.66**
November 1983	
Lie Detector Interface	**£2.45**
Microcontroller	**£2.77**
ZX Light Controller	**£5.56**
December 1983	
BBC Sideways RAM	**£6.48**
Electron A/D	**£3.78**
January 1984	
Electron I/O Port	**£3.02**
February 1984	
BBC Speech Synthesiser	**£5.89**
Electron RS432	**£3.51**
Spectrum Speech Board	**£4.18**
BBC Sideways ROM Board	**£7.13**
March 1984	
Spectrum Cassette Controller	**£2.59**
April 1984	
Commodore A/D	**£2.15**
May 1984	
Memex	**£7.55**
Spectrum Diary	**£4.26**
Centronics Buffer	**£7.41**
June 1984	
Mains Data Link (2 Boards)	**£4.72**
July 1984	
IR Data Link (2 Boards)	**£3.95**
September 1984	
Spectrum Frequency Meter	**£3.61**
October 1984	
EPROM Simulator	**£5.85**
November 1984	
Amstrad PIO	**£5.65**

HOW TO ORDER

List the boards required and add 45p post and packing charge to the total cost of the boards. Send your order with a cheque or postal order to:

E&CM PCB Service, Scriptor Court,
155 Farringdon Road, London EC1R 3AD
Telephone: 01-837 6255

Please supply the following PCBs:

....

....

Post & Packing 45p

TOTAL £

Signed .... Date ....

Name .... (please print)

Address ....

....

....

PLEASE ALLOW 28 DAYS FOR DELIVERY

Extending SuperBasic

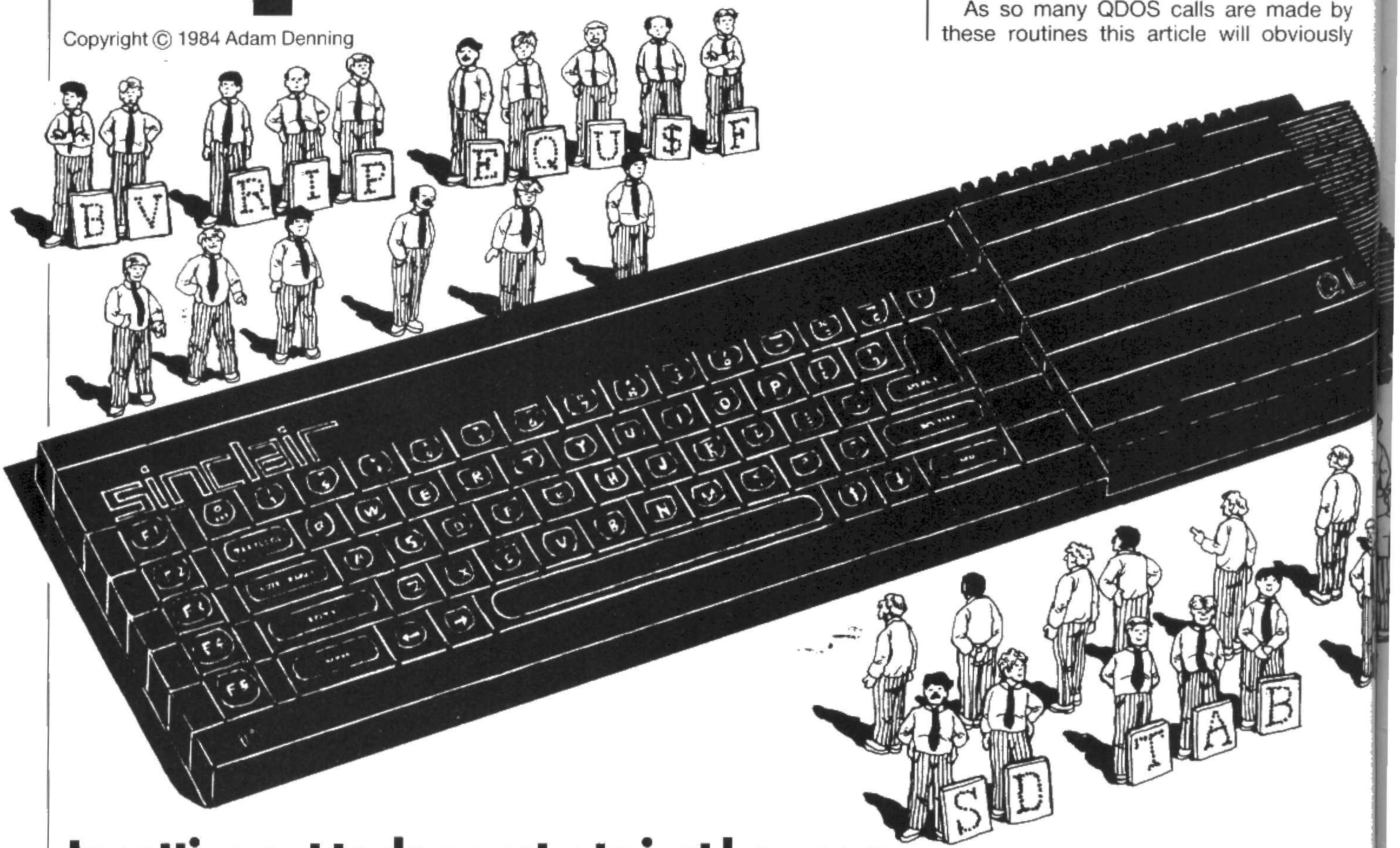


ILLUSTRATION BY MARTIN HANDFORD

In setting out to demonstrate just how easy it is to add machine code utilities to SuperBasic, Adam Denning has produced some very useful routines.

The main purpose of the additional routines is to show just how easy it is to add machine code procedures and functions to SuperBasic, but it has a rather helpful side effect as each routine is actually useful. Some give the Microdrives random access capabilities, one shows how to add user defined graphics to the machine, and the others are general purpose routines to complement those that are already present.

New QL Utilities

There are also two useful subroutines which, although not procedures in their own right, do have properties that make them very useful to have in any library of machine code programs. These are the routines DIGITS, which takes an assigned long integer and converts it to ASCII without leading spaces (and these can be added if required) and LINTOFP which converts a long integer to a floating point number, again unsigned.

Numbers are treated in three different ways on the QL – as integers, which are signed 16 bit numbers between —32768 and 32767; long integers, which are also signed but are twice as long as standard integers; and floating point, which has its own 6 byte format which will be explained later.

Numbers passed to procedures written in machine code can be collected in any of these forms, as well as strings, but return arguments can only be in integer, floating point or string format. As a number of our routines have to deal with numbers greater than the range allowed by a QL integer, we had to write a routine which converted from one format to the other.

As so many QDOS calls are made by these routines this article will obviously explain what each QDOS call does, as well as the parameters required. QDOS has two main methods of invocation – using the 68000 TRAP instruction or making use of a vector which points to the desired routine. The vectors are generally the easiest to use but the traps give access to more powerful facilities.

Firstly, here is a list of all the procedures and functions in this article, with details of how they are called from Basic, what parameters they require and details of any that they return, and what they do.

CUR#channel, state

This procedure turns the cursor attached to a screen or console device on or off depending on the value of state. If this is zero then the cursor is disabled, while any other value will turn it on. The channel must be the number with which an SCR_ or CON_ device was opened. No parameters are returned, but if less than two arguments are provided the 'bad parameter' error will occur, as it will if #channel is not a screen or console device.

UDG#channel, address

So many QL reviews have said that the machine has no user defined graphics – they're wrong. What's more, the machine can support a different set of user defined graphics for each screen or console device open, so that characters written to one channel can appear in an entirely different typeface on another. User defined graphics are not without their problems though. Each QL character is defined as being five pixels wide and nine pixels high, each character requiring nine bytes to define it. Only bits 2 to 6 are used from each of the nine bytes, the rest seeming to have something to do with colour. Each character set has the following format:

address+0

lowest ASCII value in this set

address+1

number of characters in this set minus 1 (both of these are byte locations)

address+2

start of set; this is the lowest character.

There's more to it than that, too. Each channel has two fonts attached to it, and an attempt to print a character that is not valid in font 1 will make QDOS try to print it from set 2. If it is not valid there either, then the 'chessboard' pattern you can see by typing print chr$(27) is produced.

Our UDG procedure allows you to alter the address of only the first font, as control over the second set. The first parameter is the channel whose font you are altering, and the second is the address of a font that has been laid out in the format specified. Once this procedure has been called all attempts to print to that channel will produce results dictated by the font given. An address of 0 resets the font address to that of the default font in the QL ROM.

A number of parameters other than two, or a channel which does not refer to a screen or console device will result in a 'bad parameter' error message.

As user defined graphics are fairly awkward on the QL, the author can supply a program written specifically for this purpose, allowing each character to be defined on a grid, with facilities to alter every aspect of the font. Character sets can also be saved and loaded from microdrive, and instructions for using a set once it's been defined are included. It can be purchased for £19.95 (or £14.95 if you supply your own cartridge) from:

Adam Denning
c/o Electronics & Computing Monthly
Scriptor Court
155 Farringdon Road
London EC1R 3AD.

CAT#channel, drive

Unlike the Basic procedure of the same name given last month, this procedure does rather more than just simplify the syntax or the DIR command. It catalogues a drive as usual, but places the file length (in

LISTING 1.

```
STMT            SOURCE STATEMENT
* Utilities code 1

* Extensions to SuperBasic - procedures
* and functions

* By Adam Denning
* Copyright (C) 1984 Adam Denning

          GET       'advl_header_asm'
* Operating system vectors

UT_CON    EQU       $C6
UT_SCR    EQU       $C8
UT_MTEXT  EQU       $D0
BP_INIT   EQU       $110
CA_GTLIN  EQU       $11B
BV_CHRIX  EQU       $11A
RI_EXEC   EQU       $11C

* Operating system offsets and equates

CH_LENCH  EQU       $28
BV_CHBAS  EQU       $30
BV_RIP    EQU       $58
ERR_NO    EQU       -6
ERR_BP    EQU       -15
RET_FP    EQU       2
RET_INT   EQU       3
OPEN_INX  EQU       0
OPEN_INS  EQU       1
OPEN_NEW  EQU       2
OPEN_OVR  EQU       3
OPEN_DIR  EQU       4

* Trap keys

IO_OPEN   EQU       1
IO_FBYTE  EQU       1
IO_CLOSE  EQU       2
IO_FLINE  EQU       2
IO_FSTRG  EQU       3
IO_SBYTE  EQU       5
IO_SSTRG  EQU       7
SD_CURS   EQU       $F
SD_TAB    EQU       $11
SD_FOUNT  EQU       $25
FS_POSAB  EQU       $42
FS_POSRE  EQU       $43
FS_HEADR  EQU       $47

* RI operation keys

RI_FLOAT  EQU       8
RI_ADD    EQU       $A
RI_MULT   EQU       $E

SET_UP    LEA.L     PROC_DEF,A1
          MOVE.W    BP_INIT,A2
          JSR       (A2)
          MOVEQ     #0,D0
          RTS

PROC_DEF  DC.W      5

          DC.W      CUR_PROC-*
          DC.B      3
          DC.B      'CUR'

          DC.W      PTR_PROC-*
          DC.B      3
          DC.B      'PTR'

          DC.W      UDG_PROC-*
          DC.B      3
          DC.B      'UDG'

          DC.W      BPUT_PROC-*
          DC.B      4
          DC.B      'BPUT',0

          DC.W      CAT_PROC-*
          DC.B      3
          DC.B      'CAT'

          DC.W      0
          DC.W      3

          DC.W      FN_BGET-*
          DC.B      4
          DC.B      'BGET',0

          DC.W      FN_PTR_R-*
          DC.B      5
          DC.B      'PTR_R'

          DC.W      FN_EXT-*
          DC.B      3
          DC.B      'EXT'

          DC.W      0

* Procedure to enable and disable
* cursor on a specified channel which
* is attached to a console or screen
* device. The second parameter to the
* procedure determines the cursor state
* 0 > off, NOT 0 > on.

CUR_PROC  MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_CUR
          MOVEQ     #ERR_BP,D0
          CMPI.W    #2,D3
          BNE.S     EXIT_CUR
          BSR.S     BAS_CHAN
          ADDQ.L    #4,A1
          MOVEQ     #SD_CURS,D0
          MOVE.L    0(A6,A1.L),D1
          BEQ.S     CUR_OFF
          SUBQ.B    #1,D0
CUR_OFF   MOVEQ     #-1,D3
          TRAP      #3
EXIT_CUR  RTS

* Procedure to position a file pointer
* on a specified channel attached to
* a microdrive file

PTR_PROC  MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_PTR
          MOVEQ     #ERR_BP,D0
          CMPI.W    #2,D3
          BNE.S     EXIT_PTR
          BSR.S     BAS_CHAN
          ADDQ.L    #4,A1
          MOVE.L    0(A6,A1.L),D1
          MOVEQ     #-1,D3
          MOVEQ     #FS_POSAB,D0
          TRAP      #3
EXIT_PTR  RTS

* A subroutine to return the channel ID
* (in A0) of the Basic channel which
* was #n'd to the procedure/function
* as a parameter (L_INT).

BAS_CHAN  MOVE.L    0(A6,A1.L),D0
          MOVEQ     #CH_LENCH,D1
          MULU      D1,D0
          MOVE.L    BV_CHBAS(A6),A2
          ADDA.L    D0,A2
          MOVE.L    0(A6,A2.L),A0
          RTS

* Procedure to set the fount address
* for a specified channel attached to
* a console or screen device to the
* given second parameter

UDG_PROC  MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_UDG
          MOVEQ     #ERR_BP,D0
          CMPI.W    #2,D3
          BNE.S     EXIT_UDG
          BSR.S     BAS_CHAN
          ADDQ.L    #4,A1
          MOVE.L    0(A6,A1.L),A1
          MOVEQ     #-1,D3
          MOVEQ     #0,D0
          MOVE.L    D0,A2
          MOVEQ     #SD_FOUNT,D0
          TRAP      #3
EXIT_UDG  RTS

* A procedure to send a byte to a
* specified channel

BPUT_PROC MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_BPUT
          MOVEQ     #ERR_BP,D0
          CMPI.W    #2,D3
          BNE.S     EXIT_BPUT
          BSR.S     BAS_CHAN
          ADDQ.L    #4,A1
          MOVE.L    0(A6,A1.L),D1
          MOVEQ     #-1,D3
          MOVEQ     #IO_SBYTE,D0
          TRAP      #3
EXIT_BPUT RTS

*  A function to get a byte from a
*  specified channel with infinite timeout

FN_BGET   MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_BGET
          MOVEQ     #ERR_BP,D0
          CMPI.W    #1,D3
          BNE.S     EXIT_BGET
          BSR.S     BAS_CHAN
          MOVEQ     #-1,D3
          MOVE.L    A1,-(A7)
          MOVEQ     #IO_FBYTE,D0
          TRAP      #3
          MOVE.L    (A7)+,A1
          MOVEQ     #0,D2
          MOVE.B    D1,D2
          ADDQ.L    #2,A1
          MOVE.W    D2,0(A6,A1.L)
          MOVE.L    A1,BV_RIP(A6)
          MOVEQ     #RET_INT,D4
EXIT_BGET RTS

* A function to set the file pointer
* on a specified channel attached to a
* microdrive file to the position
* given by (current_position)+2nd param.
* The function returns the new file
* pointer position, so naturally an
* offset of 0 will return the
* (unchanged) current position.

FN_PTR_R  MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_BGET
          MOVEQ     #ERR_BP,D0
          CMPI.W    #2,D3
          BNE.S     EXIT_BGET
          BSR       BAS_CHAN
          ADDQ.L    #4,A1
          MOVE.L    0(A6,A1.L),D1
          MOVEQ     #-1,D3
          ADDQ.L    #4,A1
          MOVE.L    A1,-(A7)
          MOVEQ     #FS_POSRE,D0
          TRAP      #3
          MOVE.L    (A7)+,A1
          TST.L     D0
          BNE.S     EXIT_BGET
          MOVE.L    D1,D4
          BRA       LINTOFP

* A function to return the length
* of an open microdrive file

FN_EXT    MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE.S     EXIT_BGET
          MOVEQ     #ERR_BP,D0
          CMPI.W    #1,D3
          BNE.S     EXIT_BGET
          BSR       BAS_CHAN
          ADDQ.L    #4,A1
          MOVE.L    A1,-(A7)
          MOVEQ     #FS_HEADR,D0
          LEA       CAT_BUF,A1
          MOVEQ     #-1,D3
          MOVEQ     #15,D2
          TRAP      #3
          LEA       CAT_BUF,A1
          MOVE.L    (A1),D4
          MOVE.L    (A7)+,A1
          TST.L     D0
          BEQ       LINTOFP
          RTS

* Procedure to catalogue a drive
* and give file lengths

HDR_LEN   EQU       $40

CAT_PROC  MOVE.W    CA_GTLIN,A2
          JSR       (A2)
          BNE       EXIT_CAT
          MOVEQ     #ERR_BP,D0
          CMPI.W    #2,D3
          BNE       EXIT_CAT
          BSR       BAS_CHAN
          MOVE.L    A0,-(A7)
          ADDQ.L    #4,A1
          MOVE.L    0(A6,A1.L),D1
          ORI.W     #'0',D1
          LEA.L     DEVNAME,A0
          MOVE.B    D1,5(A0)
          MOVEQ     #-1,D1
          MOVEQ     #IO_OPEN,D0
          MOVEQ     #OPEN_DIR,D3
          TRAP      #2
          TST.L     D0
          BEQ.S     CHAN_OK
          ADDQ.L    #4,A7
          RTS
CHAN_OK   MOVE.L    A0,-(A7)
          MOVEQ     #0,D5
CAT_LOOP  MOVE.L    D5,D1
          MOVEQ     #-1,D3
          MOVEQ     #FS_POSAB,D0
          MOVE.L    (A7),A0
          TRAP      #3
          TST.L     D0
          BNE       LOOP_END
          MOVEQ     #0,D4
          MOVEQ     #3,D2
GET_LEN   MOVEQ     #IO_FBYTE,D0
          TRAP      #3
          LSL.L     #8,D4
          MOVE.B    D1,D4
          DBRA      D2,GET_LEN
          SUBI.L    #HDR_LEN,D4
          BSR       DIGITS
          MOVEQ     #10,D1
```

LISTING 1 – continued

```
            MOVEQ     #FS_POSRE,D0
            TRAP      #3
            MOVEQ     #0,D4
            MOVEQ     #IO_FBYTE,D0
            TRAP      #3
            LSL.W     #8,D4
            MOVE.B    D1,D4
            MOVEQ     #IO_FBYTE,D0
            TRAP      #3
            MOVE.B    D1,D4
            TST.W     D4
            BEQ.S     NOT_DIR
            MOVE.W    D4,D2
            LEA.L     CAT_BUF,A1
            MOVE.W    D4,(A1)+
            MOVEQ     #IO_FSTRG,D0
            TRAP      #3
            LEA.L     CAT_BUF,A1
            MOVE.L    4(A7),A0
            MOVE.W    UT_MTEXT,A2
            JSR       (A2)
            MOVEQ     #IO_SBYTE,D0
            MOVEQ     #'=',D1
            MOVEQ     #-1,D3
            TRAP      #3
            MOVE.W    UT_MTEXT,A2
            LEA.L     DIG_STRG,A1
            JSR       (A2)
            MOVEQ     #IO_SBYTE,D0
            MOVEQ     #10,D1
            MOVEQ     #-1,D3
            TRAP      #3
NOT_DIR     ADDI.L    #HDR_LEN,D5
            BRA       CAT_LOOP
LOOP_END    MOVE.L    (A7)+,A0
            MOVEQ     #IO_CLOSE,D0
            TRAP      #2
            MOVE.L    (A7)+,A1
EXIT_CAT    RTS

DEVNAME     DC.W      5
            DC.B      'MDV1_',0

CAT_BUF     DS.B      38

* Routine to convert an unsigned
* long integer to ASCII
* Long integer in D4; Preserves D5.

DIGITS      MOVE.L    D4,D2
            MOVEQ     #0,D4
            MOVE.L    D5,-(A7)
            LEA.L     DIG_BUFR,A1
            LEA.L     TENTAB,A2
            MOVEQ     #0,D0
            MOVEQ     #0,D5
ONEDIG      MOVEQ     #0,D1
DIGONE      SUB.L     (A2),D2
            BCS.S     GOTDIG
            ADDQ.B    #1,D1
            BRA.S     DIGONE
GOTDIG      ADD.L     (A2)+,D2
            ADDQ.B    #1,D0
            TST.B     D1
            BNE.S     NOTLDZ
            TST.B     D5
            BNE.S     NOTLDZ
            CMPI.B    #10,D0
            BNE.S     LDZYES
NOTLDZ      ADDI.B    #'0',D1
            ADDQ.W    #1,D4
            MOVEQ     #1,D5
            MOVE.B    D1,(A1)+
LDZYES      CMPI.B    #10,D0
            BNE.S     ONEDIG
            MOVE.L    (A7)+,D5
            LEA.L     DIG_STRG,A1
            MOVE.W    D4,(A1)
            RTS

* Table of all the powers of 10
* from 10^9 to 10^0

TENTAB      DC.L      1000000000
            DC.L      100000000
            DC.L      10000000
            DC.L      1000000
            DC.L      100000
            DC.L      10000
            DC.L      1000
            DC.L      100
            DC.L      10
            DC.L      1

* Buffer for long integer to
* ASCII conversion

DIG_STRG    DS.W      1  Number of digits

* String space

DIG_BUFR    DS.B      10

* An end-of-function routine to
* convert the long integer passed in
* D4 to a floating point number
* RI stack and return to Basic
* taken care of

LINTOFP     MOVEQ     #18,D1
            MOVE.W    BV_CHRIX,A2
            JSR       (A2)
            SUBQ.L    #2,A1
            SWAP      D4
            MOVE.W    D4,D5
            MOVE.W    D4,0(A6,A1.L)
            MOVE.W    RI_EXEC,A2
            MOVEQ     #RI_FLOAT,D0
            JSR       (A2)
            TST.W     D5
            BPL.S     NOSIGN
            SUBQ.L    #6,A1
            MOVE.W    #$813,0(A6,A1.L)
            MOVE.L    #$10000000,2(A6,A1.L)
            MOVEQ     #RI_ADD,D0
            JSR       (A2)
NOSIGN      SUBQ.L    #6,A1
            MOVE.W    #$813,0(A6,A1.L)
            MOVE.L    #$10000000,2(A6,A1.L)
            MOVEQ     #RI_MULT,D0
            JSR       (A2)
            SWAP      D4
            MOVE.W    D4,D5
            SUBQ.L    #2,A1
            MOVE.W    D4,0(A6,A1.L)
            MOVEQ     #RI_FLOAT,D0
            JSR       (A2)
            TST.W     D5
            BPL.S     POSNUM
            SUBQ.L    #6,A1
            MOVE.W    #$813,0(A6,A1.L)
            MOVE.L    #$10000000,2(A6,A1.L)
            MOVEQ     #RI_ADD,D0
            JSR       (A2)
POSNUM      MOVEQ     #RI_ADD,D0
            JSR       (A2)
            MOVE.L    A1,BV_RIP(A6)
            MOVEQ     #RET_FP,D4
            RTS

            END
```

Refer to the text for a full description of the operation of each of the routines.

bytes) after each filename, like this:

```
boot=101
udg_init=3207
procs_code=872
udg_main=8712
caps=236
set_1=2000
```

It does not give the medium name or the number of free sectors on the cartridge, so it would be more useful in many applications. Error checks are made on everything except the channel to which the catalogue is to be listed – if this channel is not open then the procedure will either catalogue to channel zero or produce no listing at all. An examination of the code will show why I didn't include a check on that! It will not cause the machine to crash (or at least it hasn't yet!), so there is no reason to rewrite it.

BPUT#channel, byte

Those familiar with the BBC Micro will recognise this procedure straight away. It simply sends the specified byte to the specified channel, so that BPUT#1,65 is identical to typing PRINT#1;'A'; or even PRINT 'A';. All error checks necessary are made.

PTR#channel, position

Again very similar to the BBC Micro command of the same name, this procedure sets the sequential file pointer in a microdrive file to the absolute position specified. If this position is less than zero or greater than EOF then the 'end of file' error is produced but the pointer is placed at the requisite end of the file. Together with a couple of the other procedures and functions here, this effectively makes microdrives random access devices.

Functions

length=EXT(#channel)

This function returns the length of the file attached to the specified channel. It doesn't often make much sense if this channel is not opened to a microdrive file.

byte=BGET(#channel)

This function is identical to byte=CODE (INKEY$(#channel,—1)) and simply fetches the first character available from the specified channel and puts its ASCII value in the return expression. If the specified channel cannot support input then the requisite error message will be produced.

position=PTR_R(#channel, offset)

This is a variation on the PTR theme – the function does exactly what the PTR procedure does, except that the offset is used as a relative displacement from the current pointer position, and the value returned is the new pointer position. It therefore follows that if the offset is zero the value returned is the current position, so that if we had typed

```
PTR#3,1428
```

and then typed

```
PRINT PTR_R(#3,0)
```

we would get the number 1428 printed out – assuming that this was not past the end of file, of course. The offset can be any signed 32 bit number, with negative offsets obviously moving the pointer backwards in a file.

Before we describe the machine code in detail, it's worth making a few points about random access microdrive files. First of all, by opening an already existing file with OPEN# rather than OPEN_IN# it is possible to both read and write to that file (but Sinclair doesn't tell you that, of course!). More importantly, once you have started writing (but not reading) to a microdrive file that previously gave valid responses to the PTR_R and EXT functions, any answers returned by these functions tend to be invalid until the file is closed and then re-opened. This is certainly not the fault of the functions as they call the requisite QDOS routines, so the problem must lie with QDOS itself.

The Machine Code

The source file used to create these routines is printed below, it is in the format required by Motorola's assembler, but as this tends to follow Motorola syntax to the letter it is likely to work on all assemblers without much alteration. The important thing to notice is that it is position independent – it doesn't matter where in memory the code is situated, it will always work. Each procedure is also re-entrant, as there is no self modifying code (although the data areas are not re-entrant).

The first assembler statement is a directive to the assembler to get a file called 'header_asm'. This short source file simply contains all the QDOS definitions and equates.

We then have the code which links the procedures and functions into SuperBasic. This is very simple as all we have to do is call the routine pointed to by the BP_INIT vector, with the address of a definition block in A1. This definition block (PROC_DEF in our program) is laid out in the following format:

```
     PROC_DEF
     number of procedures           word
then for each procedure:
     offset to start of routine     word
     length of name                 byte
     characters of name             bytes
          (aligned to word boundary)
then:
     0                              word
```

number of functions word

then for each function:

as for procedures

and finally:

0 word

Calling the code at SET_UP with this table in memory is all that is required to link in these procedures and functions. Once linked, SuperBasic treats each of the new routines as though they were part of the language, and they can only be removed by resetting the machine (NEW has no effect).

"You are recommended to buy an assembler".

Each QDOS call can potentially return with an error, and this is reflected in the value of D0 on return. If D0 is non-zero then an error has occured. Returning to Basic with any other than zero value in D0 will cause the corresponding error to be produced, so we must set D0 to 0 if we want to ignore all errors.

Now we can examine each of the routines in general.

CUR_PROC

This is invoked whenever the CUR procedure is used from Basic. It first uses the CA_GTLIN vector to get all its parameters as long integers (32 bits). If this routine returns with error then we return to Basic straight away. On exit from this routine D3 holds the number of parameters collected, so if this is not 2 we leave with the 'bad parameter' error. Otherwise we call the BAS_CHAN routine which returns with the channel ID in A) of the Basic channel pointed to by the first argument on the stack. This routine will be described when we get there.

Each of the arguments is placed on what is known as the RI stack, with A1 being the RI stack pointer relative to A6. So we collect the next argument by incrementing the stack pointer by the length of a long integer and moving the argument there into register D1. If this is zero we branch to CUR_OFF, otherwise we decrement D0 which was previously loaded with SD_CURS. This is the key for TRAP #3 which switches a cursor off, while one less than this is the TRAP #3 key to turn the cursor on. We put —1 into D3 to indicate infinite timeout and then we execute the trap. Whatever error (if any) this returns is sent back to Basic so that the requisite error is reported.

PTR_PROC

This is much the same as CUR_PROC except that a different trap is used. This is TRAP #3 with the FS_POSAB key in D0, which is the routine to set the file pointer to the absolute value held in D1, again with an infinite timeout.

BAS_CHAN

Is called by a number of the routines and returns with the channel ID of the Basic channel whose # number is held as a long integer on the top of the RI stack. The channel ID is the internal number used by QDOS to identify each one channel, and is a long integer. It is found using the following formula:

(#number) * CH_LENCH + BV_CHBAS(A6) = offset from A6 of the channel definition block for that channel. CH_LENCH is the length of each channel definition block ($28 bytes) and BV_CHBAS is a SuperBasic pointer relative to A6 that holds the base address of all the Basic channel definition blocks relative to A6. This relative to A6 business is very important and often very confusing. When you're dealing with machine code from Basic, everything that you can imagine must be specified as relative to A6.

UDG_PROC

This is again very similar to the earlier procedures, differing only (essentially) in the QDOS call made. This one is TRAP #3 with SD_FOUNT in D0, and this sets the font address for the specified channel to A1 (fount 1) and A2 (fount 2).

BPUT_PROC

The same story, this time using TRAP#3 with D0 = IO_SBYTE, which sends the byte in D1 to the channel specified. As usual, the specified channel is indicated by holding its channel ID in A0.

FN_BGET

This is rather more interesting. It's our first function and so gives us the opportunity to describe the mechanism for returning a function's result to SuperBasic. The RI stack pointer must point to the return value, and it must also be stored in the Basic system variable (relative to A6!) BV_RIP. The type of the return argument must be placed in D4. As we use the IO_FBYTE trap in this routine (which collects a byte from the specified channel and corrupts A1) we must first preserve the RI stack pointer. The obvious place is on the real stack, so the MOVE.L A1,—(A7) pushed A1 onto the stack. We then execute the trap, retrieve A1 and move the byte collected (returned in D1) to the RI stack. A character never exceeds 255, so we can return the result as an integer, which is only two bytes long. This is reflected in D4.

"These extensions add greatly to the QL's power".

FN_PTR_R

Essentially the same mechanism as FN_BGET, but TRAP #3 with D0 = FS_POSRE is used. This sets the pointer on the file opened to the specified channel to a value determined by adding D1 as a signed long integer to the current pointer position. The new pointer position is returned as a long word in D1, which we move to D4 and then jump to LINTOFP to convert this to a floating point number and return to SuperBasic.

CAT_PROC

This is an extensive piece of code which is very hard to fully error-check! We first get and check the arguments as before, and we collect the Basic channel ID of the first argument using BAS_CHAN. We put this on the stack and then use TRAP #2 with IO_OPEN in D0 to open a file on the microdrive specified by the second parameter. We do this by POKEing the ASCII value of the digit passed into a device name consisting of MDVn_, loading D3 with OPEN_DIR and executing the trap. OPEN_DIR opens the directory on the specified device, interestingly enough for read and write. If no errors have occured we then branch to the main part of the routine at CHAN_OK, where we first stack the channel ID returned by IO_OPEN on top of the other channel ID.

The next stage involves reading each directory entry, which is 64 bytes long (HDR_LEN), and extracting the required information from this. The information is held in such a way that the file length is held in the first four bytes and the file name starts 10 bytes after this in the form of character count (byte) followed by characters. All we have to do is collect each file name, print it, print an '=' sign and then print the file length as an unsigned long integer. So we'll use D5 as a counter to point to each entry in the directory.

A vast loop is entered at CAT_LOOP in which we first position the pointer in the directory at the current value of D5 using FS_POSAB. If we reach end of file at this point we branch to the end of the loop, as there are no more files to be read. We then use a byte rotation to collect the long word file length in D4, with a DBRA loop (similar to the Z80's DJNZ) controlled by D2. This file length includes the length of the header, so we must subtract 64 to find the true file length. This is converted to ASCII by the DIGITS routine, and then we proceed.

FS_POSRE is used to move the directory file pointer to the start of the file name, and the number of characters in the filename is collected in D4. If this is zero the file has been deleted, so we must ignore it by jumping to NOT_DIR. Otherwise we use a highly complicated stack manipulation to print D4 characters from the directory channel to the specified listing channel, using CAT_BUF as an intermediate buffer to hold all the characters of the file name. The UT_MTEXT vector is used to actually print the filename – all this needs is the channel ID in A0 and the address of the string to be printed in A1. We then print an '=' sign followed by the digit string representing the file length. Finally, we print a line feed and increment D5 to point to the next entry in the directory. We circulate around this loop until the directory file is finished.

DIGITS

This routine converts the unsigned long

integer in D4 to ASCII, with no leading zeros or spaces. As I couldn't work out an efficient way of doing 32 bit by 32 bit division, I used the time honoured technique of progressive subtraction of powers of ten, all of which are held in RAM starting at TENTHS.

DIGITS works by subtracting a power of ten from the progressively decremented integer until the carry flag is set to indicate one subtraction too many. At this point D1 holds the number of times that power of ten occurs in our number and D0 holds the number of digits collected so far, D5 is the leading zero counter. Once the carry flag is set we must add the requisite power of ten back in once to return the number to the correct value for the next power of ten, and then a process of checking D1, D5 and D0 determines whether or not this is a leading zero. If it is, and so long as it is not the last digit in the number, it is ignored, otherwise it is added to the ASCII string in DIG_BUFR and D0 is incremented. Once the routine is finished D0 will hold the number of digits in the string, so this is transferred to the start of the string at DIG_STRG and the routine ends.

LINTOFP

This converts the long integer in D4 to a floating point number and returns it to SuperBasic as a function return value. It is rather complicated, taking note of the fact that QL floating point numbers are stored in the following format:

```
fpn = mantissa (four bytes) * (2^(exponent — $81F));
```

which means that 65536 can be represented as

```
0813 1000 0000
```

We use two utility routines here, RI_EXEC and BV_CHRIX. The latter reserves a specified number of bytes on the RI stack with the number of bytes required being held in D1. RI_EXEC performs various operations on numbers (usually floating point) on the top of the RI stack. We use three operations here: RI_FLOAT, RI_ADD and RI_MULT. The first converts a signed integer on the top of the stack to floating point and therefore requires 4 bytes of RI stack space each time it is called. The second adds the two floating point numbers at the top of stack and then the next, leaving the result at the top of the stack. It therefore reduces RI stack requirements by 6 bytes. RI_MULT multiplies two floating point numbers, again reducing RI stack requirement by 6 bytes. The process we use to convert our unsigned long integer is as follows.

A long integer can be considered as two integers, one of which is 65536 more significant than the other. As RI_FLOAT converts signed integers to floating point, we must isolate the sign of each of these two integers and add 65536 to the resultant floating point number if the original integer were negative. The complete algorithm is below.

```
long_integer = int_a * 65536 + int_b
```

bit 15 of int_a and int_b is equivalent to the signs.

```
Put int_a on stack
Convert to floating point
If bit 15 of int_a was set, stack 65536 (floating point) and add to floated int_a
Stack 65536 (floating point))
Multiply TOS with NOS
Stack int_b
Convert to floating point
If bit 15 of int_b was set, stack 65536 (floating point) and add to floated int_b
Add TOS to NOS
```

This leaves us with a floating point number on the top of the RI stack that corresponds to the long integer we originally passed in D4. As this is the function return value, we save the current value of the RI stack pointer by BV_RIP(A6), set D4 to return type float, and return to Basic.

Once this code has been assembled (presumably into a Microdrive file), the procedures and functions can be linked in by typing the following:

```
x=RESPR(length_of_assembled_code
LBYTES mdv1_<assembled_code_file_name>
CALL x
```

The ability of SuperBasic to be extended in this way makes the whole system very much more powerful than it at first appears, and you are strongly recommended to buy an assembler and learn how to do it!

WHAT'S COMING IN OUR DECEMBER ISSUE

ON SALE NOVEMBER 13TH

THE BEST SELLING COMPUTER PROJECTS MAGAZINE NOW INCORPORATING SINCLAIR PROJECTS

High Speed I/O and the BBC

Paul Beverley continues his in-depth look at the BBC micro with some examples of the techniques associated with high speed I/O.

CBM 64 Speech Synthesiser

An allophone-based unit that makes full use of the Commodore 64's programmable digital filters to reduce the component count of the project to the bare minimum.

In addition to full constructional details we show how the speech synthesiser can be programmed via a series of straightforward BASIC statements.

Floppy Turnover

No, we have not gone into the business of providing recipies! The term refers to an article that describes how you can double the amount of storage obtained from a single-sided floppy disk. Some unkind people have commented that the article is more in the spirit of 'Blue Peter' than a hi-tech computer magazine, but the scheme works and could save you a considerable sum of money.

Spectrum 64

Not a memory expansion project, but software that will, amongst other things, allow the Spectrum to display a full 64 characters per screen display line.

The QL — some important thoughts

There have been plenty of reviews of the QL which, for want of a better phrase, could be called 'rubber keyboard' reviews. We all know that the QL's keyboard is not its strongest point but there must be something more to say about the computer that is meant to be a quantum leap in computer technology.

Next month we *do* have important things to say about the QL and have discovered a number of features that may turn out to be fundamental flaws in the design.

Don't miss next month's report!

ALL AT C

C is certainly an unusual product for the Spectrum but, in Adam Denning's opinion, it is well worth buying. The language is closely linked to the UNIX OS and is probably the best language in which to write other systems packages.

Hisoft is perhaps the nearest Spectrum equivalent to Acornsoft being either the first or the best in the machine's utility field. Firstly there was Pascal, then Devpac, the belated but superior Ultrakit and now this C compiler. An unusual product for the Spectrum perhaps, but very well worth getting . . .

C is *the* systems programming language, being closely linked to the UNIX™ operating system and widely regarded as the best language in which to write compilers, operating systems and other systems packages. This is not wholly true, of course, as C's grandfather BCPL is just as useable for similar applications.

The history of C starts in Cambridge but really gets going at AT&T's Bell Laboratories somewhere in America. Dr Martin Richards had developed BCPL at Cambridge University and shown that its clarity coupled with its block structure made it ideal for assembler and compiler writing. But BCPL had one disadvantage – it had no *types*.

A type is the form in which a variable, constant or reference is stored and used in a language – for example, common variables in Basic are *floating point* (REALs in Pascal), variables ending in % are *integer*, and variables ending in $ are string variables. BCPL has none of these, everything being stored as *words*, which could arbitrarily mean anything that the programmer wished. The nearest it has to arrays is the *vector*, which is simply a consecutive group of words. Strings are held in the same way.

While this is ideal for many applications, as it ultimately makes the language more powerful, it does mean that the programmer has to be far more aware of what he is doing than he does when using Basic or Pascal. Basic, although formally typed, does allow certain 'type coercions' such as:

```
LET int_var% = float_var
```

Perhaps the most commonly used *strongly typed* language is Pascal; here it is generally illegal to assign one type to another, so the line above would be illegal. Pascal also has the problem of being less able to cope with input and output than most languages; implementations tend to get around this by cheating – extending the language.

C lies halfway between BCPL and Pascal. It has the block structures of both in that problems can be broken down into smaller parts, but it lacks the ability of Pascal to define functions within functions. And it has types but it isn't very strict about them. In fact the unwary programmer can often find that obscure type-coercion bugs are at the root of a particular evil. C is also a great deal better at handling strings than Pascal – often seen as one of Pascal's major failing points.

C is also smaller than Pascal in general, and this conciseness makes it an excellent language for the smaller system application, or the software tool. Most of the Unix utilities are written in C.

Two Americans, Brian Kernighan and Dennis Ritchie, were responsible for taking the *B* offshoot of BCPL and coming up with C. Shortly after this the just-written Unix was also converted to a mainly-C program. And with Unix, the language took off. To demonstrate a typical application, let's look at two of the Unix utilities – *lex* and *yacc.* The first is a lexical analyser which together with the (get ready for it!) LR(1). parse table created by yacc, forms the front end of a compiler for whatever language you choose. For the record, yacc stands for (rather cynically) *yet another compiler compiler,* because that's just what it is.

But no-one really expects you to write a compiler on the Spectrum. Hisoft C is probably best seen as a good way to get familiar with the language, with the added bonus that you can do useful things with it. It comes on cassette with its own integral editor, and is fully Microdrive compatible.

This version (1.0) suffers from a few QL-type 'not implemented' features, the most noticeable being the lack of the float type and the inability to save the resultant object code. These will be coming in later versions, following Hisoft's usual philosophy of constant update, but it is likely that the first things to emerge in version 1.1 (or whatever) will be the ability to compile from drive to drive and to initialise variables on declaration rather than by later assignment. Apart from those few things, Hisoft C is near enough complete – the lack of the *scanf* function is fairly insignificant as all the bits of this which are likely to be used are either in the library, can be easily written or are in the only decent book on C currently available – *The C Programming Language* by Kernighan and Ritchie.

This book is essential reading, which is rather unfortunate as it costs £16.95. The Hisoft manual refers to it constantly so it would be ill-advised not to have it to hand.

"This history of C starts in Cambridge but things get going at the Bell Laboratories".

On the reverse side of the Spectrum cassette are the two standard C files – the header for the standard library and the library itself. Together these contain numerous input and output functions, a number of definitions, store management functions, string handling functions and some 32-bit arithmetic functions. They are presented as perfectly normal source files which can be loaded into the editor or included in other compilations as necessary.

Compilation is initiated with the #include preprocessor command, which will include source in RAM or on Microdrive/cassette. If the filename is surrounded with question marks then only functions and definitions that have already been used, but not defined, will be compiled. Therefore the inclusion facility makes it easy to compile large amounts of source in one go.

However you don't have to use the editor to compile a program. When the compiler is started up source text may be entered directly into the machine, each word or symbol that is typed being compiled step by step. This method, although convenient for short programs, does have its disadvantages in that it is impossible to correct typing mistakes.

There is a third way of running C programs, too. The possibly unique #direct+ command initiates an immediate mode, whereby function invocations and statements can be typed in and executed immediately, just like interpreted Basic. On receiving the C cassette, for example, we were told to type in

```
printf("Hello Adam/n");
```

in direct mode as a demonstration. Both user-defined and built-in functions can be executed this way so obviously whole programs can also be run like this.

All C programs must have a function called *main* somewhere in the source. This function marks the place at which the compiled code will start running when the program is run. Very small programs may consist of a main definition only, but it is good practice to split larger problems into smaller, more easily defined problems, generally in the form of separately defined functions.

As our first example, here is a short and sweet program to calculate and display the Fibonnaci series between 1 and 23. A number in this series is defined as being the sum of the previous two members, with 1 and 2 both having the value of 1:

```
main()
{
      int i;
      for (i=1;i<24;++i)
            printf("%d/n",fib(i));
      }
int fib(v)
int v;
{
      if (v<=2)
            return 1;
      else
            return fib(v—1) + fib(v—2);
}
```

(The else is not really needed)

This is a very simple example, but shows up the definition of types of both variables and functions, as well as the strange (at first) C *for* loop. Here the initial value of the loop index (i=1), the loop end condition (i<24 – so the loop ends when this is false) and the loop increment (++i) are included in the enclosing brackets. In fact a for loop is just a special case of the while loop, as all the above could be coded as:

```
i=1;
while  (i<24)
       {
              printf("%d/n",fib(i));
              ++i
       }
```

"Hisoft's C is one of the most important Spectrum products launched".

Both are perfectly legal. Therefore the initialisation, end condition and loop increment can all be entirely arbitrary, with perhaps a different variable being changed or the end of loop being decided on something entirely different.

Notice too the *++i;* This is simply C's syntax for incrementing a variable. By postfixing the variable, as in I++, a slightly different effect is found, in which the value of the variable is incremented *after* it is used.

Now onto something a little more ambitious. This program will open a file on Microdrive or cassette and count all the words in it. Not exactly earth-shattering, but it does include a few of the more interesting aspects of C. One of these is the dreaded *goto* statement, included here simply to annoy the dogmatic millions. It isn't in the least necessary, of course, as it could be replaced with a *do . . . while* loop very easily!

```
#define EOF —1 /* End of file marker */
main()
{
      int c,i,count,inword;
      char s[20];
      static int *fp;
again:
      printf("/nFilename: ");
      i=0;
      while ((c=getchar())!='/n')
      }
            s[i]=c;
            ++i;
            if (i==19)
                  break;
      }
      s[i]='/0';
      fp=fopen(s,"r");
      if (fp==0)
            {
                  printf("/nFile not
                  found!/n");
                  goto again; /* Arrgghh⅛ */
            }
      count=inword=0;
      while (c=getc(fp))!=EOF)
      {
            if (i=isspace(c))
                  inword=0;
            else if (inword==0)
            {
                  inword=1;
                  ++count;
                  }
            }
            fclose(fp);
            printf("/nWords: %d/n",
            count);
}
```

This program makes no claims to be elegant and could be re-coded as more than one function invocation. The first thing it does is declare all the variables it is going to use; four as integers, one as a 20 element character array and one as a *static* integer containing a pointer. Declaring the variable as static simply means it is allocated object code space so it always present, unlike an automatic (local) variable.

The next thing is the label declaration *again:*. This is simply for use by the later goto statement. We then print out a prompt and initiate a while loop which reads a name of up to 19 characters from the keyboard. The comparison operators != and == simply mean 'not equal to' and 'equal to' respectively. The line with ++i; nn is C shorthand for 'increment i by 1'. It has two forms, prefix as above and postfix as in i++. The effects differ and each has its uses.

The *break* instruction causes the program to terminate the closest surrounding loop prematurely and in this case is used if an attempt to type more than 19 characters is made. We then convert the character array into a C string by adding a zero byte at the end ('/0') and then we pass the array to the *fopen* function. This, naturally enough, opens a file, the second argument deciding on the access type. Here it is *read.* If the file is not found (this will only happen with Microdrives) then the goto is used to repeat the process.

Otherwise we initialise *count* and *inword* to zero in the same statement (as assignment returns its value in C) and then enter a loop which reads each character from the file. If a character is a space, a newline or a tab then it will return TRUE from ispace (an in-built function) and so it musn't be counted. Otherwise we read in characters until the end of a word, incrementing count as we go.

Finally, the number of words (a fairly arbitrary definition is used) is printed out and the program finishes. With Microdrives the program works with all file types, whether they are CODE, DATA or Basic programs.

C certainly has its advantages, being concise and easy-to-write, and Hisoft's implementation for the Spectrum is an extremely good purchase for £25. In later versions it might even be used for compiling from drive to drive. It is certainly one of the most important Spectrum products launched so far.

PARLEZ PASCAL

A few years ago it required a large mainframe computer to implement a Pascal compiler. Today versions of the language are produced for a number of home computers. G. Davies begins a series of articles that explain the fundamentals of the language.

Pascal is fast becoming a popular language with home computer users. It was originally developed for the large mainframes found in universities some fifteen years ago. Nowadays, the power of these old beasts is equalled by many home computers, and it is feasible to implement Pascal on them. For instance, the ubiquitous Spectrum has a fine version from HiSoft; two versions are available for the BBC; the Dragon runs Pascal under OS9. More implementations are regularly becoming available for other home micros.

Why is Pascal so popular? The answer is simple. Pascal is a very straightforward language, easy to learn and easy to use, but nevertheless very powerful. Compilers and operating systems have been written in Pascal – a tribute to its power!

Basic is easy to learn but not so easy to use for writing large programs. Longer Basic programs tend to be convoluted affairs, lacking clarity and difficult to modify. As people wish to exploit more of the power of their computers, Pascal is a natural choice of language.

Why then, has Pascal only recently started to become popular in the home market? Again, the answer is simple. Pascal programs have traditionally been compiled whereas Basic programs are usually interpreted. Compilation involves translating a source language program (eg written in Pascal) into a computationally identical machine language program. As the machine language version will be larger, and both versions must be held in the computer at the same time – together with the compiler – a computer with capacious memory is required. Conversely, interpretation does not involve producing machine code and so uses less space.

Compilers are also a lot bigger than interpreters. For this reason, smaller home computers tend to use interpreted languages. Now that home computers have larger memories, implementing compiled languages has become feasible.

Let's now take a look at the basic structure of a Pascal program and some simple operations.

Outline program

All Pascal programs have the same basic structure, as illustrated in **Table 1**. They consist of a number of statements separated by semi-colons and terminated by a full-stop. Statements can be spread over many lines, or within a single line: it is the semi-colon which indicates the end of one statement and the start of another, not the end of a line. Notice also that there are no line numbers in a Pascal program.

The first statement of any Pascal program must be the PROGRAM statement. The word PROGRAM is a reserved word in Pascal; it cannot be used for any other purpose except as the first word in a program. As we shall see, Pascal has a number of reserved words. Throughout these articles reserved words will appear in upper case, though some implementations allow the use of lower case letters.

Immediately after the word PROGRAM must appear its name: in this case it is 'Outline'. This name is an example of an identifier: a user definable object. Pascal defines a rule for legal identifiers; they must start with a letter, followed by any number of letters or digits. Thus 'Outline' is a valid identifier, so is 'A', but '6one' is not (some implementations do not allow a mixture of upper and lower case in identifiers, or limit the length of valid identifiers).

After naming the program, we must say what it will communicate with; this will usually be the INPUT from the keyboard, and the OUTPUT to the screen. The list of 'communication channels', like all lists in Pascal, is enclosed in brackets. The PROGRAM statement is terminated with a semi-colon.

The declaration section

The PROGRAM statement has introduced our program. Now we must introduce variables. As in other languages, these hold intermediate results, answers, useful data and the like.

In Pascal, all variables must be introduced – or declared – before they are used. Basic does not allow the explicit declaration of variables, whereas Pascal requires it.

In Basic, variables are automatically declared as they are encountered, whether they are correct or not. So in a Basic program, if you mistype a variable name – say type 'Fred' as 'Frod' – no error ensues. In fact, 'Fred' would be assumed to have the value zero, which could cause a very undetectable bug. Pascal will not allow such elementary mistakes to go unnoticed. The compiler would tell you that 'Frod' has not been declared if it had been spelt incorrectly.

Not only must the variables be declared before they are used, you must also say what sort of value the variable will hold, that is, the type of the variable. There are four basic types in Pascal, but for the moment we will only consider the numeric types, INTEGER and REAL. Any variable must be of one type only, usually one of the basic types. (As we shall see, Pascal also allows more complex user defined structured types. Indeed, it is this facility that gives Pascal much of its power).

So what is a type? If we declare a variable to be of type INTEGER we are saying that it can only ever hold an integer value (a whole number, not a fraction). Thus, if we manipulate an INTEGER variable, the compiler knows it can use the fast integer arithmetic instructions of the computer.

Thus integer arithmetic is quick and easy. To store floating point numbers or large numbers, we must use a variable declared to be REAL. These can hold any number, even integers, but take longer to use because small computers do not have instruction for dealing with floating point numbers.

The variable declaration section is introduced by the reserved word VAR. We can then list the variable names – which must be valid identifiers – followed by their types. For instance:

```
VAR
   Value, Sum, Count : INTEGER ;
   Average : Real ;
```

declares four variables. 'Value', 'Sum' and 'Count' are INTEGERs and 'Average' is a REAL. (Notice how we can use sensible names for our variables as there is no restriction on identifier length, this can make programs easy to read.) Each variable declaration is a statement so must end in a semi-colon. The word VAR merely introduces the variable declaration section – it is not a statement so needs no semi-colon.

The program body

Now we reach the main program – the bit that uses the declared variables and the INPUT and OUTPUT channels to do some (hopefully) useful work. The program statements in the main body are bracketed by the reserved words BEGIN and END. Clearly, the BEGIN says to the compiler:

here is the main body of the program. Similarly the END says: thats all folks! To make it absolutely clear, the final END is followed by a full stop rather than the ubiquitous semi-colon.

What goes between the BEGIN and END bracket is the program itself. This will consist of zero or more statements implementing an algorithm. The correct choice of these statements to implement the correct choice of algorithm is the essence of programming. Let us now consider the simplest and most common statement: the assignment.

An assignment is indicated by the special symbol ':=', and it always involves an expression in the right of the symbol, and a variable to the left. The value of the expression is worked out and the variable is given – or assigned – this value. So an assignment statement is equivalent to a LET in a BASIC program. There is, however, one major difference; a consequence of the assignment of Pascal variables. An expression is considered to have a type – dependant on the types of its constituent parts – and the assigned variable must be of the same type. For instance, take the assignment

```
Fred:=Bill * Joe;
```

If 'Bill' and 'Joe' are both REALs, then this is a valid statement only if 'Fred' is also declared as REAL. However, if the expression contains both INTEGERs and REALs, it is considered to be REAL. So the assigned variable must again be REAL. It is only if 'Bill' and 'Joe' are both INTEGERs that 'Fred' can also be an INTEGER.

Type conversion can also occur: if the expression is an INTEGER, the assigned variable can be a REAL. In this case, the expression is worked out as an integer, and then converted to floating point format before the assignment.

As you might expect, the four standard arithmetic operations are available: addition, subtraction, multiplication and division. The use of types causes an interesting problem in the case of division. This is because dividing two integers will usually produce a real result. Therefore the expression

```
Bill/Joe
```

is considered to be REAL even if both the variables are INTEGERs. Should you actually require an INTEGER division, you must use the DIV operator. It is only valid between INTEGERS, so

```
Bill DIV Joe
```

gives an INTEGER result so long as 'Bill' and 'Joe' are declared as INTEGERs, otherwise a compilation error is produced. What the DIV operator actually works out is how many times 'Joe' can be subtracted from 'Bill' before the answer goes negative. So given the statements,

```
Bill:=6;
Joe:=2;
Result:=Bill DIV Joe;
```

we get a 'Result' of 3. Had 'Joe' been 4, the 'Result' would have been 1 (as 4 can only be subtracted from 6 once before the answer goes negative).

The above example shows the use of a special class of expression: the literal. This is just jargon from assembler language programmers for a simple value such as 3. As you would expect literals or even single variables may be used wherever an expression is called for.

In general, it is base practice to use literals throughout your program. If the value of the literals need to be altered, you must go through your program locating each instance of them. This can be time-consuming and error prone. Pascal provides a special class of object to help get around this problem: the constant. A variable can be declared and a value assigned to it; as it is declared as a constant, the compiler will make sure you never try to alter its value in a program. Constant declarations must come before the variable declarations. They are introduced by the reserved word CONST. For instance:

```
CONST
   Value = 100;
```

declares a constant called 'Value' which is the integer value of 100. This can now be used as if it were a variable, but as it cannot be altered should only appear on the right hand side of assignment statements. The compiler works out the type of the constant – in this case an INTEGER – from its actual value. The advantages of constants will become apparent as you write your own Pascal programs, or modify those written by other people.

Pascal compilers come with one pre-declared constant: MAXINT. This is the value of the largest number that can be stored in an INTEGER variable, and is normally related to the word-length of the processor. If at any time an INTEGER variable is used to store a value greater than MAXINT (or less than –MASINT) an error occurs and the program will stop with a suitable error message.

Now let us turn our attention to some of the other statements a program can contain. So far we have only looked at the most common statement: the assignment. There are two other classes of statement: input/output statements to transfer information into and out of the program via the communication channels; and control statements that allow the programmer to alter the order in which statements are executed.

There are two basic input/output statements in Pascal: WRITE and READ, which are similar to PRINT and INPUT in Basic. (There are some more complex and powerful statements available, but these are rarely implemented on home computers so will not be discussed any further.)

The WRITE statement writes data to the OUTPUT channel declared in the program statement, whereas the READ statement accepts data from the INPUT channel. We can write variables and text and mix the two in the same WRITE statement. For example

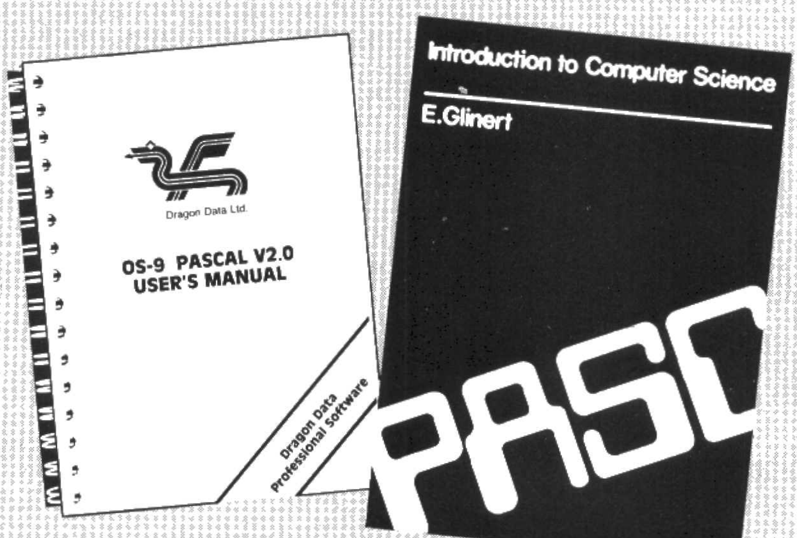
```
Fred:=25;
WRITE ('Fred =', Fred);
```

will produce the output

```
Fred=25
```

Pascal has default formats for writing variables which are usually sufficient. We shall see later how to override these defaults. A further important point is that Pascal uses a write buffer. When a WRITE statement is executed, the characters to be output are transferred to this buffer and left there. A further statement – WRITTEN – causes the contents of the buffer to be sent to the OUTPUT channel, together with a carriage return. The buffer is then cleared (but none, unfortunately not all Pascal implementations work in write this way!). This means that you can use a number of WRITE statements to build up a pretty output line, and then get it printed out with a single WRITELN.

In fact, this is such a common requirement that a short-hand version is available. Here you can specify the data to be output in the WRITELN statement rather than in a


Dragon Data Ltd.
OS-9 PASCAL V2.0
USER'S MANUAL
Dragon Data Professional Software
Introduction to Computer Science
E.Glinert
PASC

separate WRITE. Thus:

```
WRITE ('X=', X);
WRITELN;
ERITELN ('X=',X);
```

have the same effect. Note also that the text to be output – which must be between quote makers – cannot contain a carriage return itself. Carriage returns can only be produced with a WRITELN statement.

The behaviour of the READ statement is very similar, except that data values are input to variables rather then output from them. So

```
READ (A, B, C);
```

causes the variables 'A', 'B', and 'C' to have values assigned to them as read from the input channel (not from a DATA statement within the program). The declared types of the variables to be read determine how the READ is performed. If 'A' is declared as REAL, then the characters read in will be interpreted (if possible) as floating point numbers. So if the value 10 is read in even though it is an integer, the REAL value 10.0 will be assigned to 'A'.

As you might expect, these is also a read buffer and a READLN statement. The buffer contains a full line of input, and READ statements extract data from this. When a READLN is executed, the rest of the buffer is thrown away – whether it has been completely read or not – and a new line of input is put into it. (Again, though, some implementations may be slightly different.) Thus a

```
READ (A, B);
READLN;
```

reads the first two values from the buffer and then gets a new line. This can also be shortened to

```
READLN (A, B);
```

which has exactly the same effect.

There is no direct equivalent to Basic's INPUT statement, you cannot output a prompt message and read in data with the same statement. However, judicious use of WRITEs and READs can easily give the same results whilst being of more general purpose use.

Now let us consider some of the control statements available. (We will look at more of them in the next part of this series.) The simplest is the FOR statement. As in Basic, this allows the execution of a section of program for a controlled number of times. In Pascal, the controlled statement – the one we execute a number of times – can only be a single statement. This is not actually a restriction at all. For example:

```
FOR I:=1 TO 10 DO
  Count:=Count+1;
```

You can see that the FOR statement is built up from an assignment statement and the controlled statement. Notice that because there is only ever one controlled statement a NEXT to indicate the end of the loop is unnecessary. What happens is that the control variable – in the example, 'I' – is assigned an intial value, in this case 1. The control variable's value is then compared to the upper limit of the loop – here it is 10. If the variable is greater than the upper limit, we move on to the next statement in the program. Otherwise the controlled statement is executed, the controlled variable is incremented by one and compared to the upper limit again. Thus the loop repeats until it is terminated by the control variable becoming greater than the upper limit.

Pascal is very strict about the control variable. Its value must not be altered within the controlled statement – it must be a constant in that statement. It must also be a previously declared INTEGER variable, and can only be incremented by one (next month we shall see that this is not strictly true). If you wish to count down rather than up, the reserved word TO must be replaced by DOWNTO. The controlled variable will then be decremented by one rather than incremented.

As the FOR loop is built from an assignment statement, the upper and lower limits can be expressions rather than simple values. The controlled statement can be any valid program statement, even another FOR loop, thus allowing nested loops. In particular, the controlled statement can be a compound statement. This latter structure overcomes the limitation of only allowing a single controlled statement. It consists of a BEGIN END bracket around any number of statements, but is considered to be single! For instance

```
BEGIN
  Statement;
  Statement;
  Statement;
END;
```

is a valid compound statement. These statements can appear wherever one is required. Notice that, as a statement, it is terminated with a semi-colon. This distinguishes the END from that at the end of your program!

The FOR loop allows the repetition of a statement for a fixed number of times, but is fairly restrictive on the control variable. The WHILE loop gives greater flexibility; its basic form is:

```
WHILE expression1 relational operator
  expression 2 DO statement;
```

The two expressions and the relational operator form a conditional statement: one that can only have two values, TRUE or FALSE. Each expression can be any of those allowed in assignments. The relational operator is merely a type of comparison. **Table 2** gives the full range of relational operators allowed in Pascal, together with their representations. What happens is that the two expressions are compared according to the type of comparison specified by the relational operator. If the result of this comparison is true, then the controlled statement is executed and the comparisons made again forming a loop. If the comparison is false, the loop terminates. An example will make this clear:

```
WHILE A > B DO
BEGIN
  A:=A — B;
  Count:=Count + 1;
End;
```

Here our two expressions for comparison are simple variables. The relational operator is 'greater than', so if the value in 'A' is larger than that in 'B' the controlled statement is executed and the loop restarted. Notice that the controlled statement is a compound one, and that we alter the value of one of the controlling variables in the loop. (If we couldn't do this the loop would never terminate!) Assuming the value of 'Count' is zero at entry, and the other variables are delcared as INTEGERs, this loop implements 'Count:=A DIV B' when 'A' and 'B' are positive.

You should also be able to see that the FOR loop can be simulated with a WHILE, this enables restrictions inherent in the FOR loop to be overcome.

The last statement we shall consider this month is the REPEAT UNTIL loop. Often in programs we want to guarantee to execute a loop at least once, perhaps to read in a list of numbers terminated with a zero. While such a loop can be implemented with a WHILE, the REPEAT loop is a lot easier to use. In a WHILE loop, we test a condition, and execute the controlled statement for as long as the condition is true. The REPEAT loop is very similar, but the controlled statement is executed first, and then the condition tested. If the condition is false the loop is repeated. An example

```
REPEAT
  READ (Value);
  Sum:=Sum + Value;
UNTIL VALUE = 0;
```

will read the list of numbers and add them together until a zero is encountered. Notice that the reserved words REPEAT and UNTIL act as brackets, so a compound statement is not required.

This discussion of typed variables serves merely as an introduction to Pascal programming and is certainly not extensive enough to full appreciate the powers of the language. In future articles we will look at some more statements, the other two basic types and some user defined types.

TABLE 1. Outline program.

```
PROGRAM Outline (INPUT, OUTPUT);
  Declarations;
BEGIN
  Main program;
END.
```

TABLE 2. Relational operators.

Symbol	Meaning
=	Equal
<>	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

Amstrad CPC464 I/O port

Robert Penfold describes an I/O port for the computer that is set to become the Spectrum's natural successor.

Although the Amstrad CPC464 computer has a good range of ports, it lacks any form of user port. The printer port can be used as an 8-bit output plus 1 bit input, but this is rather restrictive, and means that the printer must be disconnected while the port is used for other purposes.

This simple add-on for the CPC464 overcomes the problem by using the so-called "floppy disc" port to provide two 8-bit input/output ports, and each port also has two useful handshake lines. The ports are provided by a Z80A PIO – the parallel interface adaptor in the Z80A series of peripheral chips. This has four operating modes, including one which enables each of the eight bits to be individually programmed as an input or an output. There are also input, output, and bi-directional modes. All inputs and outputs are TTL compatible.

The circuit

Very few components are used in the unit, as will be immediately apparent if you refer to the circuit diagram of **Figure 1.** The "floppy disc" port is in reality a general purpose port which provides full Z80A address, control, and data buses, together with some other useful lines including a +5 volt supply output.

The system of input/output mapping used is not the standard Z80 arrangement, where only the eight least significant address lines are decoded. Instead, input/output devices are activated by taking one or two of the eight most significant address lines low, and it is merely necessary to decode these together with the IORQ line (which goes low when any input or output circuit is accessed). The lower eight bits of the address bus are free for use if an input/output circuit has several registers and requires a number of addresses.

For external add-ons address line A10 is the important one, as it is this line going low that is used to activate external circuits. On the face of it, in this case A10 could simply be connected to the negative chip select input of IC2: the Z80A PIO. There is no need to decode it with the IORQ line as IC2 has an input terminal for this line. It also has an RD (read) input, but no WR (write) input because the write signal (in 6502 fashion) is generated internally by the lack of an active read signal.

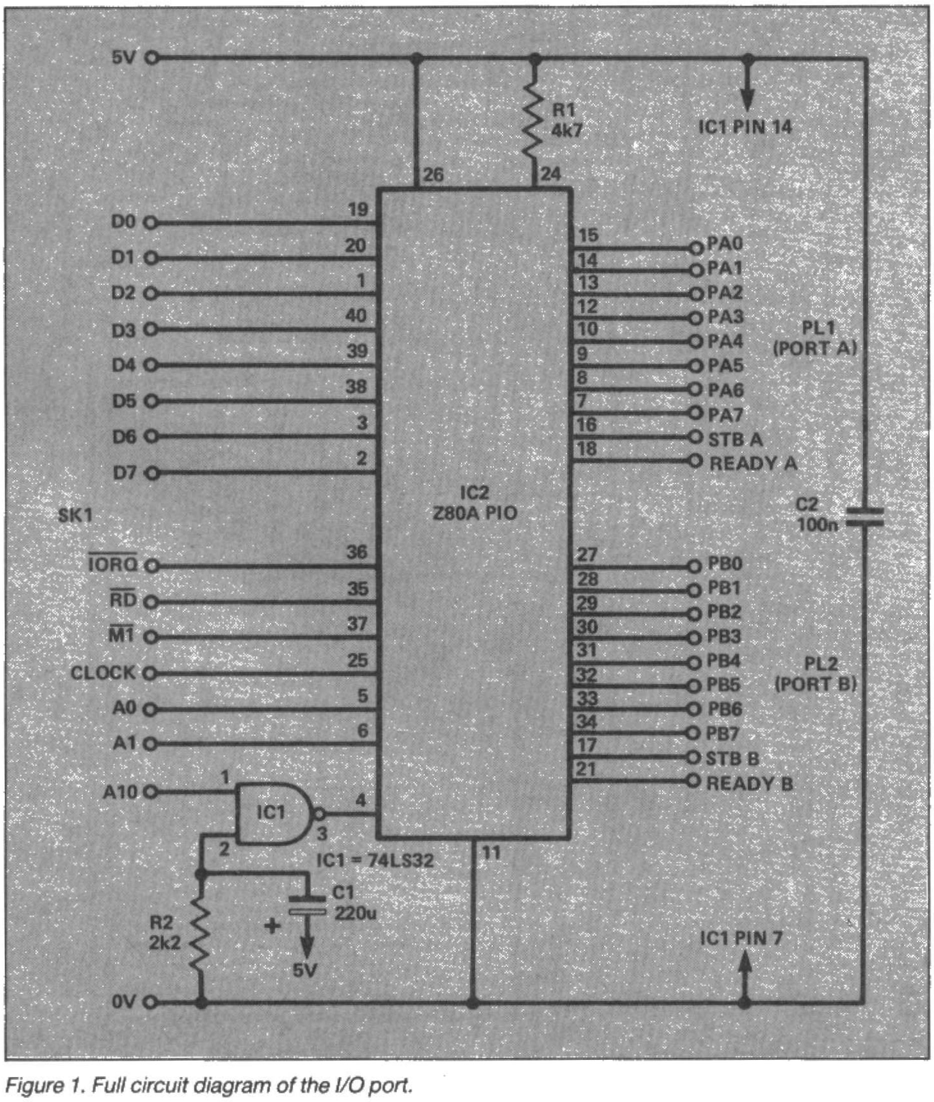


Figure 1. Full circuit diagram of the I/O port.

In practice there is a slight problem if A10 is connected directly to the chip enable input: spurious operations of the device are produced at switch-on (there is no reset input on the Z80A PIO incidentally). These spurious operations are only important in that they result in the device starting off in an unknown state, which makes programming uncertain and difficult. Also, the eight data lines of each port should normally be set as input initially, in case one or both of the ports are connected to outputs of a peripheral circuit of some kind. The spurious operations could alter this.

The simple solution to the problem is to couple A10 to the negative chip select input by way of OR gate IC1. The other input of IC1 is fed from the simple C – R timing circuit which is comprised of R2 and C1. The action of this circuit is to prevent any chip select signals from being passed through to IC2 until a second or so after switch-on, thus avoiding the unwanted spurious operations.

IC2 has inputs for the (4MHz) clock and M1 (machine cycle 1) lines. Note that as a 4MHz clock is used it is essential for IC2 to be a Z80A PIO, and not the slower Z80 PIO.

There are four internal registers, a control register and a data register for each port. Pin 5 is used to select either a control or a data register, and pin 6 is the port A/port B select input. These are fed from address lines A0 and A1 so that the four registers appear in the input/output map at the following addresses:

&F800	**Port A data**
&F801	**Port A control**
&F802	**Port B data**
&F803	**Port B control**

Each register actually appears at numerous addresses, but in practice the ones listed above are convenient to use, and safe in that they will not cause unwanted operations of internal hardware of the computer. It is important to use addresses that will not take any of the other eight most significant address lines low since this could result in unwanted operations of the internal hardware. As the Z80A microprocessor used in the CPC464 utilises input/output mapping rather than memory mapped I/O, from-BASIC data is sent to or read from the chip using the OUT instruction and the INP function, not POKE and PEEK.

The interrupt capability of the Z80A PIO is not likely to be required, and is therefore left unimplemented in this circuit. The two interrupt outputs are just ignored, and the interrupt enable input is simply tied to the 5 volt supply rail via R1.

Each port has eight data lines (PA0 to PA7 for Port A and PB0 to PB7 for Port B), plus Strobe and Ready handshake lines. The Strobe line is an input, and not as one might expect, an output which provides a negative strobe pulse each time data is written to the port. It is mainly used for latching data into the port when it is used in the input mode. The Ready line is a handshake output, and it is used to indicate that fresh data has been written to the port, or that the last byte of data has been read from the port (depending on whether the output or input mode is used). The handshake lines are described in more detail later.

Construction

The unit is built on a double-sided printed circuit board, as detailed in **Figure 2.** Start by fitting through-pins or Veropins at points where through-board connections are required. If Veropins are used, trim off the pins almost flush with the board prior to soldering them in place. Then add the other components, but use a 40-pin DIL IC socket for IC2 as this is a MOS device. Do not fit it into the socket until the project is in other respects finished, and leave it in the anti-static packaging until then. It will almost certainly be necessary to bend the pins of IC2 inwards slightly before it will fit into the socket. As this is a fairly expensive device be very careful not to buckle the pins by forcing it into place.

Connection to the floppy disc port of the computer is via a 2 by 25-way 0.1 inch pitch female edge connector. This is soldered to the two rows of pads on the board, and if the connector is a type with very long terminals it is advisable to trim them to about 10mm in length. Use plenty of solder so that the board and the connector are firmly fixed together.

Each port is taken to a 20-way IDC plug mounted on the board. Each of these has ten earth connections which provide screening between the signal leads if connections are made to the ports via the usual 20-way IDC connector and full 20-way ribbon cable. "Right-angle" type plugs are used on the prototype board, but the "straight" type will also fit onto the board without any difficulty. **Figure 3** gives details of the two ports.

To use

The board simply plugs onto the floppy disc port at the rear of the computer side uppermost. With some types of edge connector it may not fit very tightly in place, but this does not seem to give any reliability problems. Fit the board in place before turning on the computer, which should operate normally after switch-on. Turn off at once and recheck the board if the computer exhibits any form of malfunction.

Assuming all is well the port is then ready for testing. The appropriate control register is used to set each port in the desired operating mode, and the lower four bits are always set to 1 when setting the operating mode. The two most significant bits are used to select the required mode. The

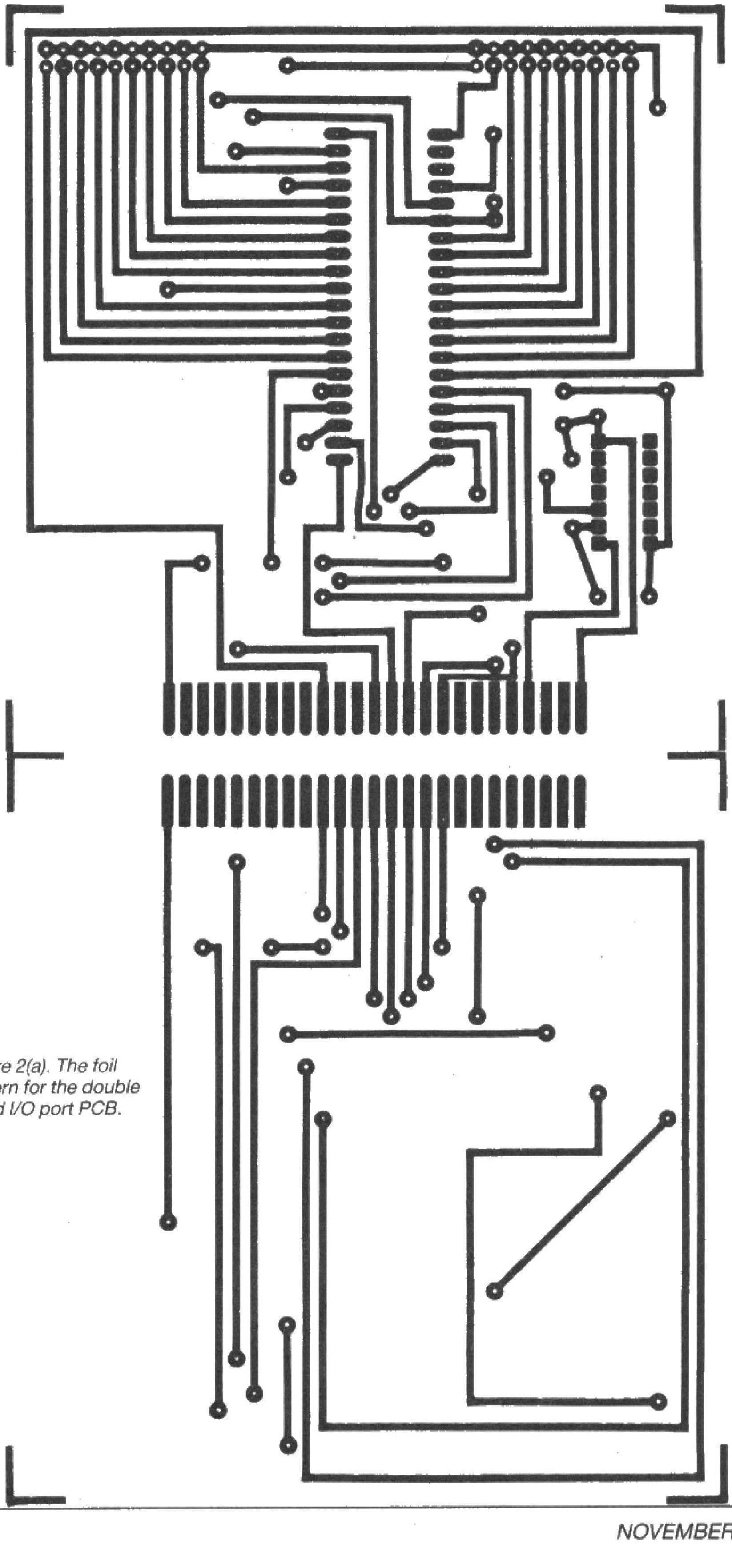


Figure 2(a). The foil pattern for the double sided I/O port PCB.

table given below shows the (decimal) number that should be written to the control register to obtain each of the four operating modes.

15	**Output mode**
79	**Input mode**
143	**Bidirectional mode**
207	**Bit mode**

The bidirectional mode is a fairly complex one which is only available on port A (port B must be set to the bit mode), and uses all four handshake lines. It is unlikely that you will need this mode, and it will not be considered here, but the Z80A PIO data sheet provides full information on this mode including timing diagrams.

When initially testing the unit it is probably best to select the output mode. For instance, the commands:

```
OUT &F801,15
OUT &F800,240
```

would set port A to the output mode, and would write 240 to this port (PA0 to PA3 low, and PA4 to PA7 high). A logic tester or a multimeter set to a low DC voltage range can be used to confirm that the outputs have latched at the correct states.

If the handshake lines are not required they can simply be ignored. If they are used, the Ready output goes low on the first negative clock edge after data has been latched onto the outputs and valid data is available. It remains low until a negative pulse is received on the Strobe input (Ready is reset as Strobe returns to the high state).

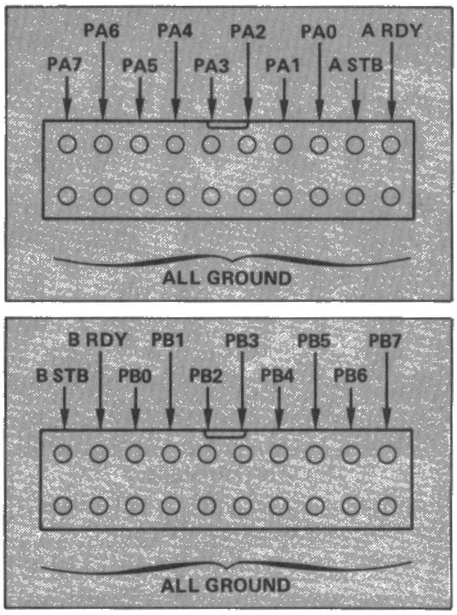


Figure 3. Pin out details of the project's two ports.

To check the input mode the following simple program can be used:

```
10  OUT &F801,79
20  PRINT INP(&F800)
30  GOTO 20
```

Line 10 sets port A in the input mode while lines 20 and 30 repeatedly read the port and print the returned values on the screen. If, for example, PA0 to PA6 are all taken low and PA7 is taken high the returned value should be 128.

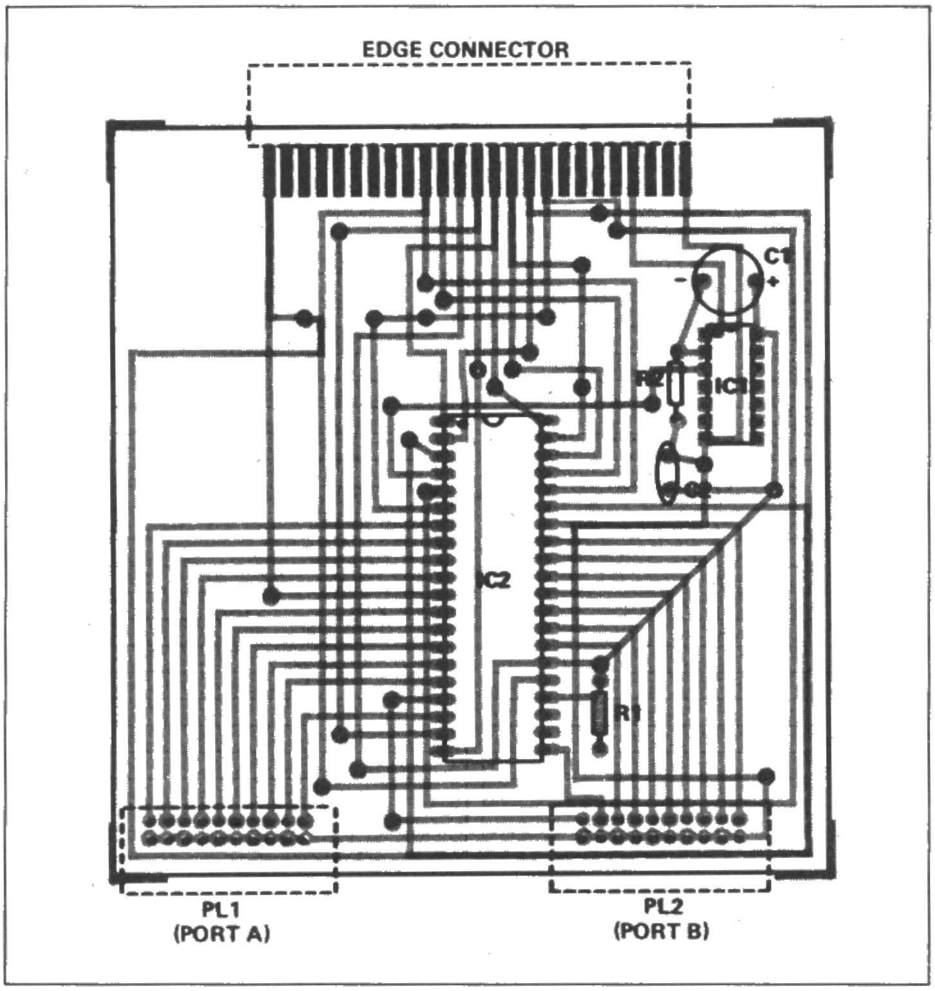


Figure 2(b). The component overlay.

Again, the handshake lines can be ignored if they are not required. When they are used a negative pulse on the Strobe input latches data into the port. This can be useful in an application such as one where a continuously converting analogue to digital convertor is driving the port. By getting the convertor to latch each new conversion into the port, reading the port always returns the result of the most recent conversion without needing to resort to any further handshaking. However, the Ready output is available if required. This goes low when data is latched into the port, and it does not go high again until the data has been read by the computer.

In the bit mode each bit of the port can be set as an input or a latching output, but the handshake lines do not operate in this mode. After writing 207 to the control register to set the bit mode the next byte to this register sets each line of the port in the required mode. Setting a bit to a 1 places the corresponding line in the input mode, while writing 0 to a bit sets the corresponding line of the port as an output. For example, the following two lines would set PA0 to PA3 as inputs, and PA4 to PA7 as outputs.

```
OUT &F801,207
OUT &F801,15
```

The CPC464's Locomotive Basic supports a bitwise AND function which can be used to mask unwanted bits when reading a port. There is also a WAIT instruction which can be used to halt a program until the specified bit of the given input port goes high or low (depending on the version of this instruction that is used). The bit mode is probably most useful in cases where the handshake lines are inadequate. One port can then be used as an 8-bit input or output while the other is used in the bit mode to provide handshake inputs and outputs.

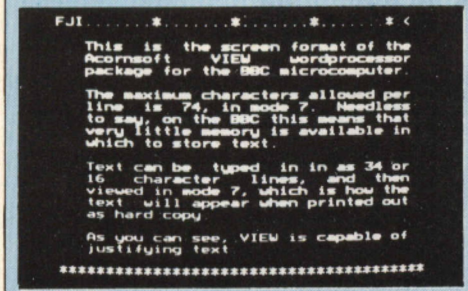
When operating the ports using machine code bear in mind that only input and output instructions in which the B register provides the upper eight bits of the address bus are applicable to the CPC464's method of input/output mapping.

PARTS LIST

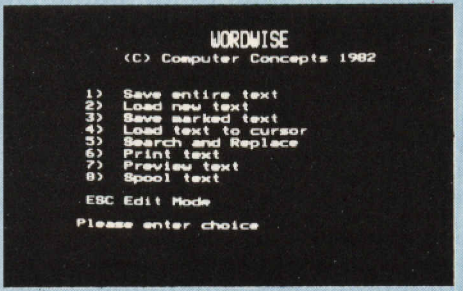
Semiconductors	
IC1	74LS32
IC2	Z80A PIO
Resistors	
R1	4k7 1/4W 5%
R2	2k2 1/4W 5%
Capacitors	
C1	220uF 10V radial elect
C2	100nF ceramic
Miscellaneous	
SK1	2 x 25-way 0.1 inch pitch edge connector
PL1,2	20-way IDC plugs

Double sided printed circuit board, Through-pins or Veropins, 40-pin DIL IC socket, 14-pin DIL IC socket.

A WORD TO THE WISE



```
FJI.......*.......*.......*......* <

     This is the screen format of the
     Acornsoft    VIEW    wordprocessor
     package for the BBC microcomputer.

     The maximum characters allowed per
     line is 74, in mode 7.  Needless
     to say, on the BBC this means that
     very little memory is available in
     which to store text.

     Text can be  typed  in in as 34 or
     16   character   lines,  and  then
     viewed in mode 7, which is how the
     text  will appear when printed out
     as hard copy

     As you can see, VIEW is capable of
     justifying text

**************************************
```



```
                WORDWISE
      (C) Computer Concepts 1982

   1)   Save entire text
   2)   Load new text
   3)   Save marked text
   4)   Load text to cursor
   5)   Search and Replace
   6)   Print text
   7)   Preview text
   8)   Spool text

   ESC Edit Mode

  Please enter choice
```

'The only word processor most people like is the one they are used to'.

The validity of this oft-quoted statement is demonstrated by the reluctance of users to part with a limited and dated system for the bells and whistles of a modern word processor.

The reason for this isn't too difficult to discern: any software package, no matter how user friendly, will take time to master before its full power is realised. Word processors in particular require extensive use of keyboard commands.

That said, it is all the more important to make the right choice from the numerous packages available for home micros. To make the decision easier, *E&CM* has reviewed the best wordprocessors available for five leading home computers: the BBC micro, Spectrum, QL, and Commodore 64.

The term 'wordprocessor' is applied to any software that allows the user to manipulate text files within a computer. This can range from the very basic systems – text editors, which edit and amend text but do little else, to powerful systems which include full text handling facilities along with spell checkers and mailmerge functions.

Needless to say the more powerful the system the more you pay for the privilege. It is therefore important to assess your requirements to avoid buying facilities that will be left unused. For example, short sharp letters to the bank manager do not require spell checkers or mailmergers. But a club secretary using his home computer to send out membership renewal notices will see mailmerge as essential. Equally a professional author will require a spellchecker.

To assess a representative sample of the many wordprocessors available, or panel of reviewers were given the task of producing a standard letter. The letter was designed to bring out all of the most important functions of a wordprocessor, and each package was assessed in terms of the ease of use of the commands.

Another important part of the reviewers' brief was to comment on the quality of on-screen help facilities and documentation available. Even the best wordprocessor, if accompanied by a poorly written manual, will be difficult and frustrating to use.

Software is by no means the only consideration: of vital importance is the volatile memory capacity and keyboard of the computer you own or intend to buy. As you will see from our review of Tasword 2, it can be a soul destroying experience typing out large quantities of text on a Spectrum. But again, for those letters to the bank manager it is perfectly adequate.

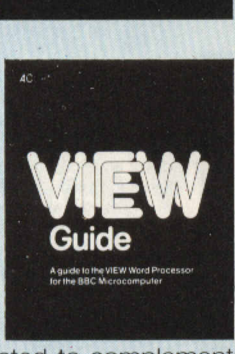
Similarly there are excellent text editors available for the BBC micro which suffer because the memory can only accommodate the entry of a lengthy file in 40 column mode.

At the other extreme it is possible to run a full professional wordprocessing system under CPM on a BBC micro with Z80 second processor. The cost of such a configuration would, with printer, screen and disc drives, cost in the order of £1650. To do the job properly, this is the sort of outlay required, and we review three popular packages for the BBC with second processor for that reason.

Compromises are available: The Commodore 64 with wordcraft fits into this catagory; these systems will prove adequate for most home wordprocessing requirements.

Further hardware considerations will be whether to use disc drives or cassette recorder (no problem here for QL owners), colour or mono monitor, and of course – if word processing it to be more than an academic exercise – which printer? On this last point readers are referred to the printer review in the September 1984 issue of *E&CM*, but the ideal situation would be to see the printer demonstrated in action with the wordprocessor before buying.

VIEW


VIEW
Guide
A guide to the VIEW Word Processor for the BBC Microcomputer

View is Acornsoft's word processing system for the BBC microcomputer. In contrast with the likes of Wordstar and Stylograph, this word processor was written specifically for the BBC computer and as such can be expected to complement the hardware configuration of the computer in a way that the more general packages written to run under an operating system rather than a particular computer cannot hope to do.

As may be expected, View makes full use of the BBC computer's function keys, and many of the word processors commands are called into action by use of one of the computer's ten function keys.

View is a ROM based package and as such there is no need to wait while the system software is loaded from either disk or tape; to enter View it is only necessary to issue the command

*WORD

upon powering up the computer. View features two distinct modes of operation, the command and text modes. When first entered View is in the command mode, to switch to text mode the command NEW must be issued. At this point text may be entered in the usual free form without having to pay attention to the screen, wrap round ensures that the system will automatically move to a new line when the current line is full. While in text mode a number of what are termed immediate commands may be issued. These commands have an automatic effect on the displayed text and most are obtained via the function keys as mentioned above. Immediate commands include cursor control, from single space/line movements to top (or bottom) of text commands. Also provided are the usual insertion and deletion facilities both of single characters and of complete lines. The system incorporates a full range of formatting commands and provides a command for producing justified text output.

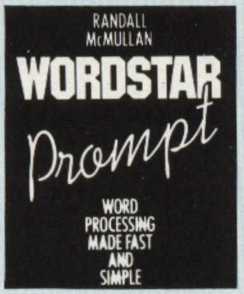
Another group of commands are termed stored commands, as their name suggests these have no immediate effect on the display but are stored in the text file to be used to control the printed output of the file. This group of commands includes those that set the margins, the page length (default 66 lines), and a facility for centreing lines of text. View provides a comprehensive set of commands for the creation of headers and footers to appear on each page of the print out.

View offers a basic form of mailmerge facility by way of its macro command. This allows the user to create a block of text which may at a later stage be recalled by use of a simple two letter command. A macro allows up to ten symbols to be incorporated within it, these being by information from a series of parameter files at the time of printing. Thus it is possible to create, for example, standard letters, and to fill in details of name and address from parameter files in order to create personalised correspondence.

View provides many other facilities including 26 number registers which may, with the exception of two predefined counters for pages and lines, be allocated as required and a command that can supply a word count of either an entire file or a marked block of text.

View manages to provide a comprehensive range of functions which should meet the requirements of the majority of home micro users. Its main drawback is its inability to display on screen an exact representation of the appearance of the printed output. This is a failing of many micro-based word processors however, and there is a school of thought that says that however accurate the on-screen representation of a file is, when the text is first printed out there will be some aspect of it that is not as expected and that most users will produce at least one hard copy of their text before being satisfied with the final result.

WORDSTAR



As one of the most widely-used CP/M based wordprocessing packages on the market, WordStar is paradoxically one of the most loved and hated of those available. However, despite objections that it is over-complicated and dated, it still has over a million users and so must have some advantages over its rivals. We put the package through its paces on a BBC micro with Upgrade Technology's Z80 2nd processor on CP/M.

Having waded through realms of profuse documentation it is just about possible to learn how to use WordStar despite apparent attempts to hide or obscure every command. The manuals are unnecessarily detailed and do not provide any short cuts for the first time user, generally serving only to confuse.

However, of far greater value, are the help menus which can be easily accessed at any time whether editing or creating text.

Text is created in files which, depending on their form are either document or non-document files. With a Mailmerge option (a facility which often comes supplied with WordStar but not with the BBC package) files of text can be merged with data files and so documents such as personalised letters may be produced.

The size of a file will depend on disk size and there is no limitation on file length apart from the amount of machine memory available. The more full a disk becomes the longer it may take to be saved. An 80 column screen will give the user a near-accurate impression of what the file will look like when printed out; the printer and bold commands appear on screen but the result is not too far off the final hard copy version.

WordStar will automatically wordwrap and justify each line to the same length. This process can be altered by a justification command followed by one which will make text ragged right. Text can be centred quickly and margins and tabs set up with the options from the on-screen help menu.

On a CP/M based system small insertions and deletions may be made with an insert option and there are several commands to make the moving of the cursor more rapid thus speeding up any changes necessary. In this way the cursor can be moved backwards or forwards, whole words or lines at a time, much more rapidly than with the specified keys.

Blocks of text may also be moved around or read by marking the text and shifting it to specified places which can be within the same file, to another one on the same disk or even to a different disk by a CB/M command process called PIP. Text is marked at the beginning and end of the piece and the block written across. Documents can be merged in this way providing that a marker is used within the same files or a new file set up for text to be transferred to.

WordStar also provides facilities for finding and replacing parts of the file being edited. The search may be done through the file globally or forwards and backwards, any number of times. Items can be replaced with or without asking for confirmation. Search and replace will be helpful when, for example, a word has been mis-spelt throughout a document.

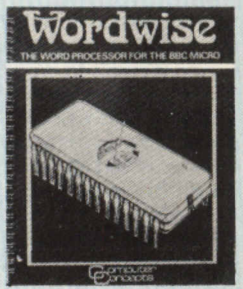
The optional package SpellStar provides a good dictionary for the user which is easy to access and update with a 20,000 basic word count. Errors can be double-checked, words may be added and even a supplementary dictionary with additional words may be created if needed.

Printer commands concerning paper size and margins are prefixed by a . in the margin before the command. Thus any stray fullstops at the beginning of the line will result in a ? opposite it on screen. Top and bottom margins, automatic headings and optional page numbers can all be dealt with by this procedure. Any special effects required for the printer such as underlining, bold, and striking out are prefixed by the command appearing on screen and extra effects can be created depending on the type of printer.

There are certain differences between a BBC version of WordStar and those configured for business machines. For example, when establishing page design and using different line spacing, the BBC code line has to go back to the menu to change over whereas a business user can do this with the on-screen menu. Scrolling facilities and screen handling are unsatisfactory. The BBC uses only about 20 lines of screen and leaves at least 5 lines of clear screen at the bottom, there is no obvious reason for this other than memory shortage. It is also unfortunate that the mailmerge and spellcheck facilities do not come along with the package. Apart from this, though, there is little difference between the BBC micro implementation and business versions of WordStar.

Once the commands have been learnt, WordStar is a pleasant system to work with. What takes the time is continual cross-referencing which is not necessary once the user is familiar with the package. Once mastered it could be particularly useful for the small businessman.

WORDWISE



Many BBC micro users will be familiar with Wordwise, which – with Acornsoft's VIEW – is one of the most popular ROM based BBC wordprocessors.

However Computer Concepts have now made available a new version which incorporates embedded OS (star) commands, load and run execution routines, printer selection during output, and which is compatible with the Aries RAM expansion board.

Like any BBC wordprocessor used without a Z80 and extra memory, Wordwise text is viewed on the screen in 40-column mode. To preview the text as it will be printed (in 80-columns) the user must return to the menu and choose the preview option. This restriction makes the creation and design of documents difficult; in particular a letter, with the sender's address and date ranged to the right.

All Wordwise commands (other than those contained within the menu which are concerned with saving and loading text and search and replace) are accessed via the BBC's 10 function keys. Eight of the keys constitute a full command: F1 sets text input mode (insert or overlay); F4 marks the beginning and end of a block of text to be moved or deleted; F5 moves the cursor to a point in the text; F6 sets the word count; F7 sets a line below which text should be deleted; F8 and F9 delete and move marked text; and F10 copies marked text.

All other commands are accessed via F2 and F3. F2 signifies the start of an 'embedded command'. Any text preceded by an embedded command appears on the screen coloured green. F3 ends the embedded command. The commands are all contained within the manual (there is no on-screen help) and control such things as justification, margins, tabulation, centreing etc.

Specific printer control commands (ie machine code instructions) can be entered within the text if preceded by an embedded command plus OC.

Using the embedded OS commands available in version 1.2 it is possible to access facilities available in other ROMs, such as printer utilities.

Wordwise is not the easiest wordprocessor to use, certainly not until the various commands are familiar. Starting from scratch is a difficult process and marked text has the habit of going to the last place you would expect. Another difficulty, caused by the hardware, is the limited size of available memory. This is relieved somewhat in version 1.2 if used with an expansion board; this allows the user to enter text to the memory limit but still preview it in 80 column mode. Unfortunately the extra memory can only be used for previewing text, not for holding larger text files.

WORDCRAFT

Wordcraft is another example of a word processor that is an adaptation of an existing package rather than an original piece of software. While this approach has obvious cost benefits as far as the authors of Wordcraft are concerned, the limitations of the CBM 64's hardware means that, in some cases, the performance of the system as a whole leaves something to be desired. Having said that though the shortcomings of Wordcraft are in the main shared by the majority of wordprocessors designed for use with home micros.

Wordcraft is designed for use with Commodore's immensely successful '64 micro together with either a parallel or serial printer. It's well known that Commodore micros are not the easiest of computers to hook up to a printer, and anybody wishing to 'play it safe' is well advised to stick to one of the range of printers that the company market for use with the 64.

The software supplied for review was disk based and, in order to use the package it requires that the Wordcraft dongle is connected to control port 2, in addition to the standard computer, monitor, printer, disk drive set up. Wordcraft is loaded with the usual command of the form:

```
load"*",8
```

after the software has loaded, which thanks to Commodore's rather quaint serial-based disk data transfer system can take rather longer than you might expect, the familiar READY prompt will appear. Typing RUN will now cause Wordcraft to announce itself in suitably modest fashion by spelling its name in large letters across the screen. When this display is over-hitting the space bar will start the session proper.

At this stage a list of printers which the system will support will be displayed, each with a corresponding letter (as with the other reviews in this article, we shall assume that the problems of configuring a printer have been dealt with) pressing the letter key corresponding to the printer in use will put Wordcraft into one of its two major modes of operation; ie the COMMAND mode. In this mode, and indeed throughout any Wordcraft session, the first five lines of the screen are used to display information about the status of the system. These include the name of the document being worked on and the current date in the first line along with the present position of the cursor (col. and line) plus the page number in line 2. Line 3 displays information as to the current mode in which Wordcraft is being used and displays various prompts or error messages that may be produced during a session. Line 4 is again used to display error messages and echoes commands entered from the keyboard while line 5 indicates the current setting of the ruler which will discussed later.

It is from the command mode that the information regarding the configuration of the system is entered. These include the setting of the page width of the final document; the maximum width allowed is 117 columns although the screen is only capable of displaying 40 of these (this is one of the areas in which the hardware of the '64 lets Wordcraft down) and setting various other parameters such as page length and naming the document.

Text is entered from the aptly named TYPE mode, the move from COMMAND to TYPE mode being accomplished by pressing the stop key. In type mode text is entered in free form, word wrap being provided by Wordcraft. In order to terminate a line at the end of a paragraph it is necessary to either press the function 7 key or to hit the CBM key followed by return, simply hitting return alone will have no effect.

Having keyed in a text file the document may be saved by issuing the appropriate command from the command mode of the system. Before leaving the description of text entry mention should be made of the ruler facility provided by Wordcraft. This is called up from the command mode and allows the user to establish margins and to set or clear tab stops. Both margins and

WORDPROCESSORS

COMPUTER	TITLE	PRICE	MEDIUM	SIZE	DISTRIBUTOR	FEATURES
ATARI	ATARIWRITER	65.00	CR	16K	ATARI INTERNATIONAL	CARTRIDGE BASED WORDPROCESSOR/WILL SAVE DOCUMENTS ON DISC OR CASSETTE/INC MERGE AND STANDARD LETTERS
BBC	WORDPERFECT	8.95/ 11.95	C/D	32K	DOCTOR SOFT	40 TO 80 COL WORDPROCESSOR/INC JUSTIFICATION, WRAPAROUND, CUT AND PASTE
BBC/CPM	WORDSTAR	—				
BBC	VIEW	59.80	ROM	32K	ACORNSOFT	PROFESSIONAL WORDPROCESSOR IN ALL RESPECTS OTHER THAN PRINTER UTILITIES, WHICH ARE LIMITED
BBC	VIEW PRINTER DRIVER	9.95	C	32K	ACORNSOFT	MAKES UP FOR VIEW'S INADEQUACIES BY DRIVING WIDE RANGE OF PRINTERS INC EPSON MX/FX80, DIABLO, QUME ETC BBC
BBC	BEEB PEN	45.00	D	32K	BRAINTECH	
BBC	WORDPRO	10.50	C/D	32K	IJK SOFTWARE	WP WRITTEN FOR CASSETTE AND FEATURING JUSTIFICATION/INSERT/DELETE ETC WITH PRINTER DRIVERS FOR EPSON AND SEIKOSHA. DISC VERSION COSTS EXTRA
BBC	WORDPROCESSOR	19.95	C	32K	GEMINI	CASSETTE BASED PACKAGE WITH 6000 CHARACTER MEMORY
BBC	WORDSWORTH	17.25/ 19.50	C/D	32K	IAN COPESTAKE	CHEAP TEXT EDITOR
BBC	ALPHABETA	28.50	C/D	32K	H+H SOFTWARE	TEXT EDITOR – NOT SO CHEAP
BBC	WORDWISE	46.00	ROM	32K	COMPUTER CONCEPTS	WITH VIEW, THE MARKET LEADER IN BBC WORDPROCESSING/A COMPETENT 40/80 COLUMN TEXT EDITOR
CBM64	EASYSCRIPT	75.00	D	64K	CBM	'PROFESSIONAL' WORDPROCESSOR WITH OPTIONAL SPELLCHECK 'EASYSPELL'
CBM64	THE WORD	50.00	D	64K	IMPEX	
CBM64	VIZAWRITE	79.95	C/D	64K	VISA SOFTWARE	FULL WORDPROCESSOR/INCLUDES 'SPELLCHECK' 'VISASPELL' FOR EXTRA £20.00
CBM64	WORDCRAFT 40	89.95	CR	64K	AUDIOGENIC	CBM64 VERSION OF THE BEST SELLING PET WORDPROCESSOR
CBM64	PAPERCLIP	89.00	D	64K	KOBRA	FULLY FEATURED/SIMPLE TO USE WORDPROCESSOR WITH NUMERICS
DRAGON 32	WORD PROCESSOR	17.25	C	32K	JUNIPER COMPUTING	INCLUDES ALL NECESSARY FEATURES FOR BASIC TEXT EDITING/COMPATIBLE WITH MOST PRINTERS
DRAGON 32	TELEWRITER	—	C	32K	MICRODEAL	SEARCH AND REPLACE/EDIT/DELETE/COPY/ PRINTER MARGIN DRIVERS
DRAGON 64	STYLOGRAPH	79.95	D	64K	DRAGON DATA	RUNS UNDER OS9/40 COL WORDPROCESSOR WITH FULL FACILITIES
MTX500/512	PROPEN 32	40.25	C	32K	ELSTREE	STANDARD CASSETTE BASED WORDPROCESSOR
NEWBRAIN	WORD PROCESSOR	40.25	C	32K	ELSTREE	VERSION OF MTX PROPEN
ORIC	AUTHOR	14.50	C	16K	TANSOFT	INCLUDES WORD-WRAP AND WORD COUNT/ TADO/CENTREING
ORIC	WORD PROCESSOR	17.25	C	48K	JUNIPER	MENU DRIVEN WP WITH USUAL EDITING FUNCTIONS INC JUSTIFICATION
QL	QUILL	FREE	MD	128K	SINCLAIR	SOPHISTICATED WP BUNDLED FREE WITH QL/EXTENSIVE HELP FACILITIES/LIMITED IMPORT-EXPORT FACILITIES TO DATABASE
SHARP MZ80/700	WDPRO	69.95	D		KUMA	FULL SCREEN TEXT EDITING AND GLOBAL COPY/ UNDERLINING/TABS
SPECTRUM	THE WORD PROCESSOR	9.95	C	48K	MICROL	SIMPLE TEXT EDITOR BUT INCLUDES JUSTIFICATION AND CENTREING
SPECTRUM	SHIFTY	7.50	C	48K	WORKFORCE	SIDEWAYS PRINTING ON ZX PRINTER
SPECTRUM	SPECTEXT	9.95	C	48K	MCGRAW-HILL	ACCOMPANIES BY EXPLANATORY BOOK
SPECTRUM	TASWORD	7.95	C	48K	TASMAN	BEST SELLING SPECTRUM WORD PROCESSOR/ IMPROVED UPON BY...
SPECTRUM	TASWORD TWO	13.90	C	48K	TASMAN	
VIC20	VICWRITER	19.95	C/D	11K	CBM	STANDARD TEXT EDITING FUNCTIONS
VIC20	WORDCRAFT20	89.95	CR	UX	AUDIOGENIC	ALSO AVAILABLE WITH 8K EXTRA MEMORY FOR FOR ADDITIONAL £35.00/INC CENTRONICS PRINTER INTERFACE
ZX81	TASWORD	6.50	C	16K	TASMAN	SIMILAR TO SPECTRUM TASWORD ONE

tabs may be altered at any part of the text and it is thus possible to achieve some fairly imaginative layouts for the final printout of the system.

Wordcraft supports many of the functions expected of wordprocessors including block shifts and the various forms of the find, search and replace facilities. All of these actions are instigated by straightforward commands that make use of single letter commands following depression of the CBM key.

Wordcraft is a well-written piece of software that should meet the word processing requirements of the majority of CBM 64 users. The manual that accompanies the software is presented in a clear and easy to comprehend fashion that will allow most users to put the package to immediate use. The status information presented at the top of the screen page is a useful innovation that allows the current state of the system to be readily determined. Overall the package offers good value for money.

TASWORD

Given the nature of the keyboard on the Spectrum any serious attempt at word processing without modification on this machine cannot really be contemplated. However, if you must use a Spectrum, then Tasword II is a good wordprocessing package as any providing as many facilities as a price of £13.90 might be expected to proffer.

The manual which accompanies the software is clearly presented and provides precise instructions for each command and this is adequately supported by the on-screen help menus. It is a pity, though, that the documentation has no easy reference index which makes the task of finding specific commands difficult.

Text files of up to 320 lines may be created and a TV screen holds 22 of these lines with 64 characters upon each one thus making a fair total of characters per file. There is a special command in the extended mode which will enlarge the characters so that only 32 characters may be fitted on a line. The file name can be up to 10 characters long and the file can be saved reasonably rapidly. Many of the commands are in extended mode which is achieved by caps shift and symbol shift plus the relevant command. Through this the user can scroll up or down, format, set margins, mark blocks and find and replace text. The automatic justification is turned off via this mode otherwise Tasword will wordwrap.

There is a facility to merge files as long as this is done one after another and the total is not more than 320 lines. Printer commands are included in the extended mode and there are additional commands for the Epson FX80 printer which opens up a whole new list of features for the Spectrum.

Tasword is a easy-to-use wordprocessing system and, despite the drawbacks of the Spectrum keyboard, provides adequate text facilities. With the use of a special keyboard, like the Lo Profile Professional or that supplied by DK Tronics, and a reasonable printer together with the new double density disk interface by Watford Electronics and disk drives, the system becomes much more feasible as the software is adequate.

QUILL

To make word-processing a practical rather than a nightmarish proposition on a home computer, four vital accessories are required: software; screen; disc drive(s); printer. The total cost including computer ranges from £800 to £1500 – or did, until the Sinclair QL came along.

The QL has an excellent word processing package – Quill – included within its purchase price, and because it incorporates microdrives no disc drives are necessary.

Our happy 8/32-bit computer owner is now faced with two severe drawbacks (the catch). Firstly, the keyboard of his £400 computer is hopelessly inadequate: it is much too slow and stiff for touch typing. Secondly, microdrives are considerably slower than disc drives, and the tapes are prone to corruption and breakage.

Quill has two qualities which place it head and shoulders above the Wordstar/Wordcraft generation of wordprocessors: firstly, a simple but extensive command structure accessed through an on-screen control area; secondly, a true 'what-you-see-is-what-you-get' screen display.

The text as it appears on the screen is exactly as it will be printed out. This facility includes the use of coloured characters to indicate bold text, on-screen underlining, and of course 80-column character width (which in itself is unusual on a home computer unstrung with second processors, expansion cards etc.).

The control area occupies the top four lines of the screen, and is divided into seven sections. The centre section merely shows how to begin a new paragraph (by pressing return!) and to delete. The other sections show which of the five function keys to press to enter commands and utilities, ie F1 for help, F2 for prompts, F3 commands, F4 page style, F5 glossary.

At the foot of the screen is a status area displaying mode, number of words written, page number and document name.

Before examining the finer points of Quillmanship, it is worth taking a quick look at the commands available via the two command screens accessed by function key 3; some are self explanatory, others will have to be defined: they are Copy, Design, Erase, Footer, Goto, Header, Justify, Load, Margins, Print, Quit, Save, Tabs, Files, Hyphenate, Merge, Page, Replace, Search, and Zap.

Briefly, the Copy command is used for moving or copying text from one place to another; any number of copies of a section of text can be made.

Design is a page formatting facility which sets up line spacing, lower and upper margins, lines per page, start page, type density, display width (for those using 40 column TV screens), and characters per inch.

Files has five sub-options: backup (very necessary with microdrives); import (to insert files from the database, spreadsheet, or graphics package); rename; delete.

Justify has three options: ragged right; justified right and left; and centred.

Within the margin command is a facility to reform all or part of the text to a new margin width. This facility is also available within the Justify command.

Tabs include an excellent range of options, providing numeric (decimal) tabulation, and centred and left or right ranged text – very useful for designing tabular documents.

Search and Replace is used to search the document for a particular word, and then gives the user the option of leaving the word as it is or replacing it with a previously specified string of characters. This facility is most useful for correcting a document in which one word is consistently mis-spelled.

Finally Zap deletes a whole document instead of saving it to the Microdrive.

Glossary is an unusual facility accessed via F5. It can be used to store standard oft-used phrases (eg 'Yours faithfully', 'I apologise for overdrawing my bank account'). Each phrase is printed to the screen by pressing Shift F5 followed by a designatory letter.

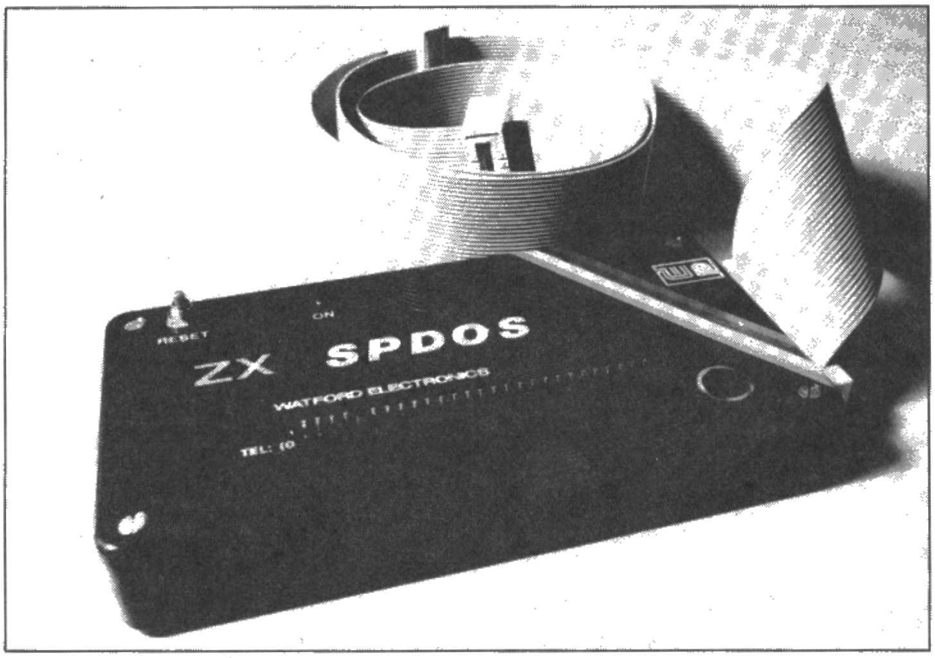
Despite the wealth of commands available in Quill (and there are others which we have no space to describe) they are very easy to access, rarely requiring more than two or three key depressions.

SUPPLIERS

ACORNSOFT	0223 316039
ATARI INTERNATIONAL	0753 33344
AUIOGENIC	0734 586334
BRAINTECH	01 997 8986
CBM	0753 74111
COMPUTER CONCEPTS	0442 63933
DOCTOR SOFT	0903 206076
DRAGON DATA	0656 744700
ELSTREE	01 953 6921
GEMINI	0395 265 165
H+H SOFTWARE	0928 65566
IAN COPESTAKE	048 674755
IJK SOFTWARE	0253 55282
JUNIPER COMPUTING	0662 2689
KOBRA	0491 572512
KUMA	07357 4335
MCGRAW-HILL	0628 23432
MICRODEAL	0726 3456
SINCLEAR RESEARCH	0276 685311
TANSOFT	02205 2261
TASMAN	0532 438301
VISA SOFTWARE	0634 813780
WORKFORCE	

ZX SPDOS

Peter Luke reviews a new disk interface for the Spectrum that offers excellent performance at a competitive price.



Without doubt the Spectrum is an extremely versatile computer capable of far more than simply the playing of computer games and there is a wealth of software written for it which one might describe as of a serious nature. This issue of *E&CM* makes mention of two packages which serve to illustrate this point. Hisoft's PASCAL is an obvious example of a package which is designed to appeal to the thinking computer owner and Tasword is designed to make the most of the Spectrum's power. In the case of Tasword, however, one of the drawbacks is that the Spectrum does not feature a disk interface as part of its specification. That is not to say that Tasword cannot be used very effectively with a cassette-based system but that anybody wishing to get the most from a word processor will sooner or later appreciate that a disk drive is an essential add-on.

There have to date been a number of companies who have designed and marketed Spectrum disk interfaces but the latest system to be released offers features that make it one of the most powerful and flexible of designs.

The SPDOS disk interface was designed by Abbeydale Designers Ltd and is to be marketed by Watford Electronics, whose name is well known in the home computer market. To date Watford have concentrated their efforst on the BBC micro and the launch of SPDOS represents an important move into other areas of this industry.

The hardware of the interface consists of a black plastic case which plugs into the expansion port at the rear of the computer. The unit feeds through all the lines of the expansion port so that other interfaces may be used in conjunction with the system. This capability is essential in the case of a disk interface unit as there will almost certainly be a requirement to use the system in conjunction with a printer which might be either the ZX model or, via an interface, with a standard Centronics or RS232 unit.

Another important part of the SPDOS system is the system disk. As well as providing the comprehensive set of utility software, it also contains a number of useful applications packages including the Tasword word processing system mentioned above. This will be looked at later.

The interface is designed for use with any Shugart compatible drive that is capable of double density operation. It makes no difference whether the drives are of the established 5¼" variety or of the newer 3 or 3½" designs. A multi way ribbon cable connects SPDOS to the disk drive (or drives) and will operate with between one and four drive units. Although operation with one drive is possible and the system software makes allowance for this, it is for more convenient in the majority of applications to work with at least two drives.

Powering up

When powering up the Spectrum with the disk interface connected, the drive designated as drive 1 will exhibit a brief amount of activity as the operating system is loaded into RAM before the screen displays a copyright message. The first time the system is used it is essential to make a back-up copy of the system disk.

This involves two distinct stages. Firstly the system tracks are copied onto the blank disk and then all copyable files are moved across to the back-up disk. Note the use of the words "all copyable files". In an interesting approach to the problems of software piracy, the copying process will back-up all the utilities of the master disk with the exception of the code responsible for the formatting of a new disk. This effectively means that any copies of the master disk are of little use to third parties as there will be no means of preparing disks for use with the SPDOS system.

Formatting

The formatting process is called into action by the 'copysys' command and the form of this command is:

```
PRINT #4: LOAD"copysys": PRINT 1
```

This results in the system displaying a menu asking about the characteristics of the drives in use with the system. When all the questions have been answered the blank disk will be formatted. Upon completion of the formatting process, the system will return to Sinclair BASIC, to re-enter the SPDOS system it is necessary to press the reset button on the interface, this will cause the system software to be reloaded into

TABLE 1. SPDOS Command Summary.

```
CAT
CAT :PRINT d
CAT :PRINT d, "string"
CAT :PRINT d, "string";
CLEAR :PRINT m1,m2
CLOSE #n
ERASE "filename"
ERASE "filename": PRINT d
FORMAT "diskname": PRINT i,j,k,l
INKEY$ #n
INPUT #n;var1;var2;var3 etc
LOAD "filename"
LOAD "filename": PRINT d
LOAD "filename" filetype
```

RAM. Backing up the system disk is completed by typing the command:

```
PRINT #4: MOVE"",""::PRINT 1,1
```

at the keyboard. Before going any further some explanation of the structure of the two commands so far discussed is probably in order.

All SPDOS commands are prefaced by the command:

```
PRINT#4.
```

This is followed by a number of parameters depending on the particular utility to be invoked. Those of you familiar with the concept of data streams used within the Spectrum will recognise that this command assigns stream four to the disk system. Initialisation of this stream occurs when the SPDOS is booted but it is important that stream four is not closed during operation with disks as this will effectively disable the system and could cause corruption of data.

Typing the following command:

```
PRINT#4: CAT
```

will display the contents of the system disk. Files may be of five different types as indicated by a file specifier. Those available include basic, for basic programs; bytes, for machine code programs; $data, for string arrays; ndata, for numeric arrays and sequ, for sequential access files.

The CAT command will also show that SPDOS achieves a massive 800K of storage although requirements of the operating system will mean that maximum user space is a little less than this, but not by much.

```
MERGE "filename"
MERGE "filename": PRINT d
MOVE "file1","file2"
MOVE "file1","file2": PRINT d
MOVE "file1","file2": PRINT d;
MOVE "file1","file2": PRINT d1,d2
MOVE "file1","": PRINT d1,d2
MOVE "","":PRINT d1,d2
OPEN #n,"filename"
OPEN #n, "filename": PRINT d
PRINT #n;var1'var2'var3 etc
SAVE "filename"
SAVE "filename" LINE m
SAVE "filename" filetype
```

Loading and saving

This process is achieved with the straightforward commands LOAD and SAVE both input after the initial PRINT# command. The save command supports all the Sinclair data formats and conventions including the option of supplying line numbers for auto running with the CODExx,xx command. The verify facility is not included since verification is an automatic part of the SAVE algorithm. SPDOS also allows screens to be saved for later display; the speed at which a saved screen can be retireved is about 20 times faster than the time taken with a tape system and is fast enough for some crude but effective animated sequences to be produced.

There is not room to detail the full list of commands supported by the SPDOS interface but **Table 1** provides a full list from which it will be seen that the system implements a full range of standard DOS commands.

Overlay techniques

One very powerful aspect of the SPDOS system is its ability to use overlay techniques within Basic programs. For those of you unfamiliar with the concept of overlays, a brief explanation should help to show the advantages of the idea. Overlays help overcome the limitations of RAM space that all home computers suffer from by only loading into working memory those elements of a program that are required for immediate use. Studying the sequence in which lines of a BASIC (or indeed a machine program) are executed will reveal that in a large percentage of the time, the program will be accessing program statements that are either in sequence or are only a few lines apart. This is immediately apparent if you think of the operation of a typical program that will spend most of its time either executing lines in turn, or looping round a few lines of a FOR NEXT loop. The exceptions to this rule are branches caused by GOTO statements and those invoked by the IF THEN construct, although often in the latter case the branch is again only a few lines.

This information allows the programmer to manage available memory by only loading small sections of a program from disk according to the requirements of the program. It is thus possible to run extremely large programs on computers with a comparitively small amount of RAM (these techniques find applications in many of the larger adventure programs).

Sinclair BASIC does not provide a command for clearing the lines of a program between those specified and this in order to implement the overlay concept, SPDOS provide the command:

```
CLEAR : PRINT m1,m2
```

where m1 and m2 are the lines between which lines are deleted. Using this command together with the auto number facility allows blocks of program to be swapped between RAM and disk in order to provide what to the uninitiated must seem like a vast amount of working memory.

SPDOS, incidentally, itself makes use of overlays in order to keep the amount of RAM required to support its operation to the minimum.

Other highlights of the system include the ability to support a generous number of files per disk and an extremely flexible wild card facility that can be very useful when trying to retrieve files whose name you have, at least in some part, forgotten, although of course *E&CM* readers would never suffer from this problem!

Manual support

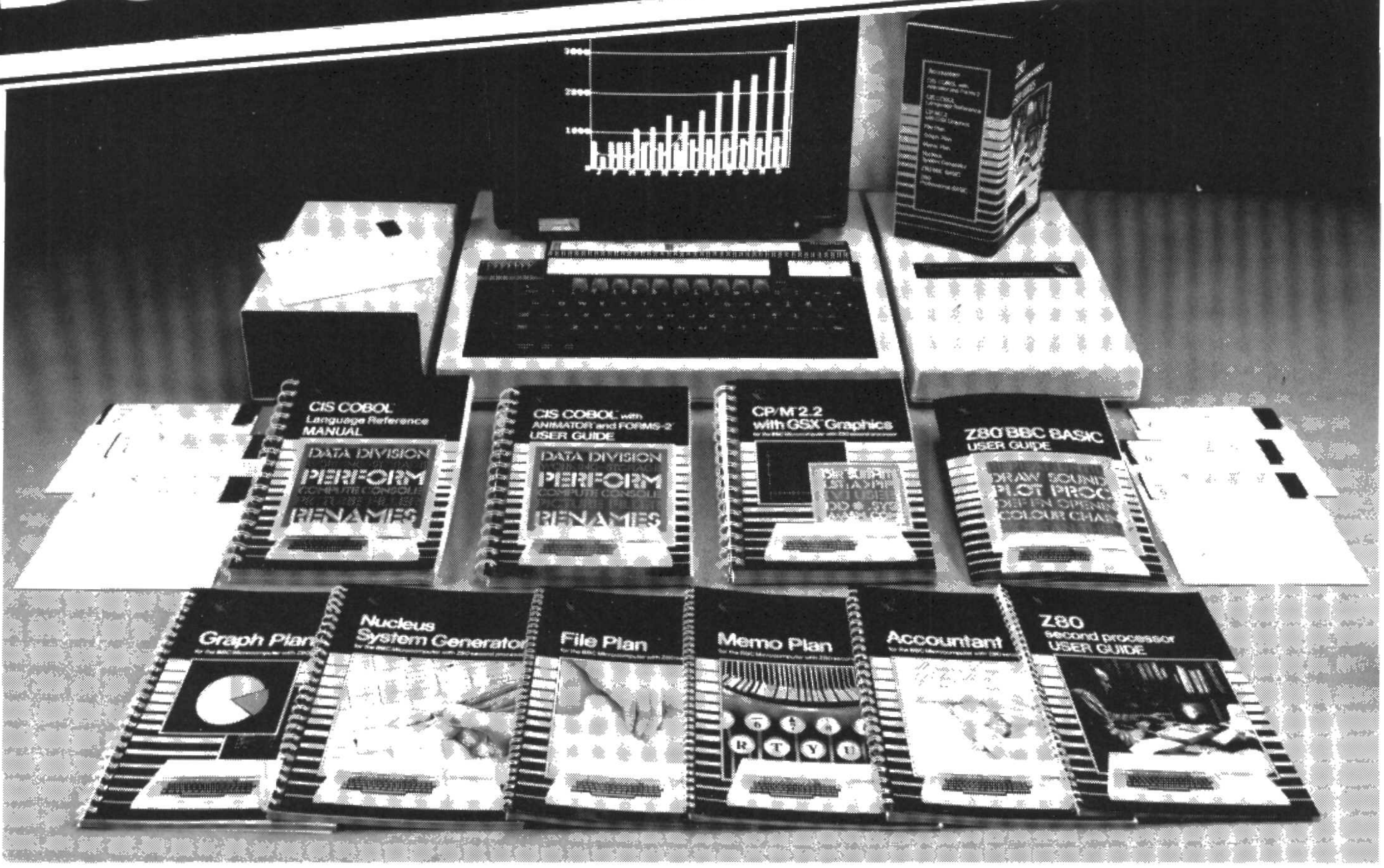
The quality of the documentation accompanying the interface was excellent. Each of the system commands was explained in sufficient detail that a novice to the tricks of using a disk system could soon feel at home. A welcome touch was the fact that the user was not just told to do something but was told why things were done in a particular way. The technical reference section at the back of the manual provided a great deal of background information as to the operation of the interface.

Before concluding, some mention should be made of the software supplied with SPDOS. This, as we have said above includes the excellent Tasword word processor and a basic form of mailing list program. This can be used for the production of simple mailing lists but the main benefit will come from studying the listing in order to learn from the techniques used by the program. In the case of Tasword, however, the user is presented with a fully fledged wordprocessor that makes full use of the capabilities of SPDOS. The software makes provision for the loading of cassette-based files so any of you with Tasword data on cassette tape will have no trouble upgrading to the disk system.

SPDOS is a well designed disk operating system that manages to achieve a high level of integration with the standard Spectrum BASIC. The use of the PRINT# to pass commands to the system is to be preferred to the USR form that is adopted in some of the other disk interfaces that have been produced for the Spectrum. One minor point to note is that, for reasons associated with the way in which the Spectrum's address lines are decoded, SPDOS is not compatible with Sinclair's Interface 1. Since the main purpose of this unit is to interface to the micro drives however, this is not likely to be a problem.

There will be many people who are considering upgrading their Spectrum to one of the newer micros that have a built-in provision for both printer and disk interfaces. SPDOS now means that anyone considering this move can give a new lease of life to their computer. By adding an additional keyboard, a printer interface, and the SPDOS system, you will have a set up that will give most 8-bit micro systems a run for their money.

Beeb Upgrading


CIS COBOL Language Reference MANUAL
CIS COBOL with ANIMATOR and FORMS-2 USER GUIDE
CP/M 2.2 with GSX Graphics
Z80 BBC BASIC USER GUIDE
Graph Plan
Nucleus System Generator
File Plan
Memo Plan
Accountant
Z80 second processor USER GUIDE

The Second Processor

Adam Denning and William Owen review three of the second processors designed for operation with the BBC Model B computer.

DATA SHEET

Product: ACORN Z80
Machine: BBC Model B
Price: £399
Supplier: Acorn Computers

Acorn's Z80 second processor for the BBC Micro is the long awaited 'official' version, and was released way behind schedule a couple of months ago. Its price has just been increased by £100 to £399, so it's a good time to examine the system and see if it's really value for money.

The main point of adding a Z80 processor is the inherent capability of supporting the CP/M operating system with its wealth of software. But CP/M is considered to be out of date and unfriendly.

The situation isn't helped by the numerous versions of CP/M available – it is now supported on processor other than the 8080/Z80 in various guises such as CP/M 68K, Concurrent CP/M and CP/M 86. Software is CP/M's only strong point and this advantage is slipping away fast.

Acorn supplies CP/M 2.2 with its second processor, the most common version of the system in use on Z80 systems, and to back up Acorn's belief in the OS the company bundles a whole load of software with it.

Apart from Professional Basic and a respected version of COBOL, the famed BBC Basic has been converted to take advantage of the Z80. This version is known as BBC Basic Z80, and is near enough identical to Basic 2 with the obvious exception that the built-in assembler produces Z80 machine code rather than 6502.

If you like programming in Basic then sure, this is as good as any, but it isn't portable. That's why Acorn also provides Professional Basic, which a number of CP/M business programs are written in.

CP/M was originally written for the antiquated 8080 processor and this is reflected in the assembler and monitor, both of which can cope only with 8080 code – not too much use on the far more powerful Z80. CP/M's editor (ED) is infamous, and rightly so. Anyone thinking of developing programs with it should give up now.

No self respecting programmer would be seen dead on a system providing this veritable dearth of facilities. To this end the rest of the bundled software is aimed fairly and squarely at the small business user, who will get a substandard word processor, a spreadsheet which makes QL Abacus look like Framework, and a business graphics package of stunningly boring quality.

The inclusion of Nucleus, the acclaimed program and report generator is Acorn's one saving grace. It is easily the most useful package in the bundle, along with its sister package called Accountant. There are a couple of other programs too, but isn't that always the story. Quality not quantity, Acorn.

The (very) small businessman would undoubtedly find this package useful, but the price (£400 for the BBC, £400 for the 2nd processor, £300 for the printer, £250 for the monitor and £500 for the disc drives) can hardly be called competitive. Shop around.

DATA SHEET

Product: ACORN 6502

Machine: BBC Model B

Price: £195.50

Supplier: Acorn Computers

With a second processor fitted, the job of the BBC becomes exactly that of an I/O processor, handling keyboard, parallel and serial outputs, text and graphics, disc drives etc. The second processor takes over the standard work of running languages and applications programs. Communication between the two processors is handled via the Tube: a fast software interface which permits high speed communication, and running any BBC program which adheres to the Tube's software protocols.

The result is a faster computer with much more memory (an additional 64K). The improvements are particularly noticeable in graphics applications, and it is no coincidence that Acorn's first software project for the second 6502 was Bitstik – a computer aided design package with utilities equal to those used by professional designers. It is a pity that Acorn left out the suitable plotter software to realise the potential power of the system.

Setting up the 6502 involves a small palace revolution in the BBC's (hereafter referred to as the I/O processor) sideways ROM sockets. Out goes the DFS, out goes the Basic, and out goes the NFS ROM. In their place is inserted a new Basic, known as Hi-Basic, and a new disc/Econet filing system, the DNFS; both ROMs are supplied with the 6502.

Once this game of musical chairs is complete the I/O processor can be switched on. The 6502 is then switched on and the I/O processor reset by control/break, upon which the existence of the 6502 is recognised with the screen message:

```
Acorn TUBE 6502 64K
Acorn DFS
BASIC
>
```

The programmer now has at his fingertips *up to* 44K user memory in Basic within the 6502, 50% faster execution speed, and can use any screen mode without worrying about the memory available.

Character fonts are automatically exploded and keyboard characters can be immediately redefined. Languages can be stored on cassette or disc, as well as accessed from paged ROM.

It may be uncharacteristic of Acorn to encourage the copying of ROMs, but under strict copyright instructions they permit users to do just that on page 20 of the user guide. Now, however, is the time to remind readers again that ROM software, like any other, will only squeeze through the Tube if it adheres to the correct protocols; unfortunately most ROM based software available for the BBC does not.

Manufacturers such as Computer Concepts are currently attempting to rewrite 6502 compatibility into their ROM software. This is not always possible because the ROMs to not have the vacant space on which to exhaustively re-write existing code. ROM code is generally not relocatable and can therefore operate only at the same location on the 6502 as it does on the I/O processor. Computer Concepts give the example of the transfer of Basic, which goes to location &B000 in 6502 RAM, leaving approximately 10K of memory above Basic, which is therefore unuseable.

6502 users have also found difficulties utilising the full 44K of RAM said to be available in Basic. In certain situations the memory available in Mode 7 appears to be limited, and Hi-Basic rarely makes more than 20K of extra memory available. Further, while the manual states that the memories of the I/O processor and 6502 are distinguishable, this only applies in machine code.

Hi-Basic makes a number of extra commands available to the user. The extensions and alterations include the following: ABS, a unary minus operator; COUNT, which counts the number of characters printed using PRINT since the last new line; ELSE, an extension of the ON . . GOTO/GOSUB to allow ELSE to be used inside procedures or functions; EVAL, which is used to evaluate strings; INPUT, an improved syntax for the entry of more than one string or variable; INSTR, another extension which returns the value of 0 if a longer string is searched for inside a shorter one ON ERROR, cannot accommodate all line numbers; OPENUP and OPENIN, the former now does the job of the latter, which itself has a new meaning – to open a file for read only operations.

DATA SHEET

Product: upGRADE Z80

Machine: BBC Model B

Price: £299

Supplier: upGRADE

Like the Acorn Z80, the main feature of the upGRADE second processor is its ability to run CP/M applications software. WordStar, Microsoft Basic, and a number of upGRADE's own business packages are available for use on the BBC with this processor board.

It is here that the similarity with the Acorn device ends, for upGRADE has used a novel solution to the problem of connecting up the Z80 to the I/O processor, and have added a number of extra features.

Instead of using the Tube interface, the Z80 is connected via the BBC's RS423 serial port. This decision has three major effects: firstly, the Z80 avoids the software bugs present in the Tube interface; secondly, speed is reduced, but the loss is minimised by the use of some clever software techniques (shortened OS calls) and floppy disc handling is speeded up by a 9600 baud bus within the Z80; thirdly, if you have a serial printer you cannot use it in conjunction with this processor because the RS423 is already occupied, unless the optional serial I/O board (price £25.00) is fitted in the Z80 expansion slot.

A further unusual feature is the disc controller fitted into the Z80's two expansion slots. A number of other expansion boards are available, including a 64K RAM board which acts as a silicon disc, recognised by CP/M as drive P (CP/M can handle up to 16 drives so the silicon disc occupies the highest of these).

Other expansion boards available include an IEEE488 interface, simple Centronics interface, PIO board, real time clock, tracker ball and interface, and a 192K RAM expansion board. There is also an 8" floppy disc controller, and a hard disc option for up to a 110 Mbytes Winchester. As a general rule only two expansion boards can be used on the Z80 simultaneously.

Operation of the upGRADE is simple; it uses standard CP/M 64 with no frills attached and has excellent documentation with a full explanation of CP/M and its commands. The manual treats the user as a novice to CP/M but avoids the pitfalls of oversimplicity. There is even an explanation of the way in which data is stored on a disc. upGRADE recommend the purchase of an advanced CP/M manual for anyone who is unfamiliar with the operating system.

Once booted up, a status report of the Z80 appears on the screen. This informs the user which expansion cards are present and have been recognised by the I/O processor. The first task is to format a blank disc in order to back up the system disc and applications software. This was our first problem. It proved impossible to format a disc without obtaining bad sector reports although this was probably because the system disc was corrupt. Numerous attempts at formatting were made, including trying to format an 80 track disc as a 40 track. Eventually we gave up, using the vacant sectors on the WordStar disc to store data. This upGRADE implementation of WordStar proved to be the real thing, with no significant changes to the command structure and no facilities lost; it was also quick.

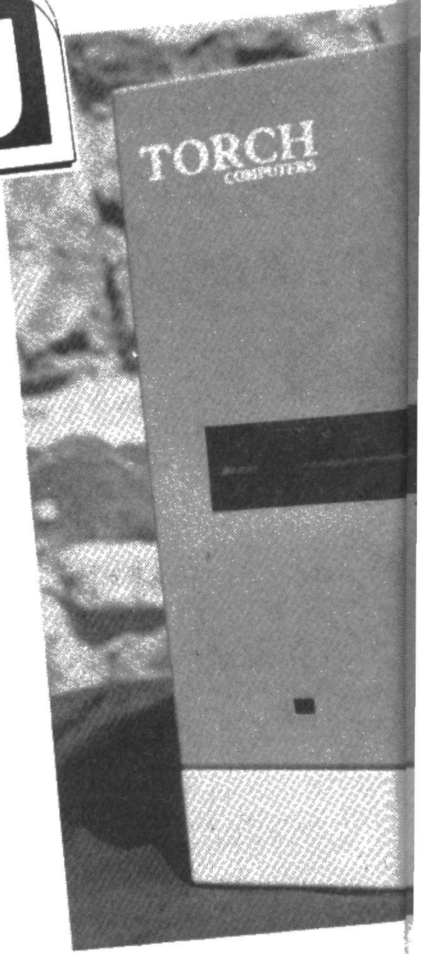
The upGRADE Z80 is a well-proven product, tested thoroughly in use on computers other than the BBC; its design is simple and uncomplicated. The crashes and failures to implement CP/M commands were therefore surprising. The disc drives used to conduct the review were also well-proven and unlikely to be at fault, so it is probable that the discs were to blame.

The two expansion slots and boards available to fit into them make this version a more useful, flexible system and all round better bet than Acorn's.

Beeb Upgrading

UNICORN 2nd PROCESSOR

Paying nearly £3000 for a second processor may seem to be a case of the tail wagging the dog. Wait until you see the Unicorn's spec. sheet though. Mike James reports.



Put simply the Unicorn is a second processor for the BBC Micro but once you hear its specifications it is difficult not to think of the BBC Micro as playing the secondary role! The Unicorn offers the BBC user an entry into the 16/32 bit world of the 68000 processor running UNIX III for far less than the price of a completely new system.

This is not to say that the Unicorn is in the same price bracket as the BBC Micro – at £2895 excluding VAT it is definitely in its own class when it comes to BBC second processors! Even if you consider the Unicorn unattainable in view of its price, you may still find some of the details of the way the Unicorn works with the BBC Micro interesting.

Torch, the manufacturers, are well known for their Z80 second processor for the BBC Micro and now that they have been taken over by Acorn it will be interesting to see what happens to those product lines which compete directly with Acorn products. However, the Unicorn and its associated hardware is quite safe in that it is unique to Torch and quite an asset as far as Acorn is concerned.

Solid hardware

The Unicorn is a remarkably small steel box considering how much it contains. Most of the space is taken by a pair of disc drives, one double sided 400K 5" floppy and one 20M 5" Winchester drive. The floppy disc drive is intended to be used for software and data interchange and for backing up files from the Winchester disc. The Winchester disc itself is, of course, the main storage device for the system. The large capacity offered by Winchester discs is not the only reason for using them in conjunction with advanced processors like the 68000; they are also very fast.

The physical construction of the Unicorn is such that the two discs are stacked one on top of the other with the Winchester at the bottom of the column to give the extra weight needed for stability. On top of the disc drives are two small PCBs holding the electronics for the Winchester interface and at the very top of the stack is a familiar switch mode power supply.

The main logic board is too large to be mounted in this way and it is accommodated at the side of the case at right angles to the disc drives and their associated electronics. This board is fairly simple, holding 256K of RAM in the form of banks of 64K chips, a Z80 processor and a 68000 processor. Apart from an EPROM holding system software that's all there is to the processor part of the Unicorn, and this suggests that if it wasn't for the Winchester disc the price of the system would be a lot less.

Overall the quality of construction is very good and the way that the disc drives and PCBs have been mounted has produced a good looking and very compact system.

"The main board is fairly simple, holding 256K of RAM in the form of banks of 64K chips, Z80 processor and a 68000 processor".

BBC interfacing

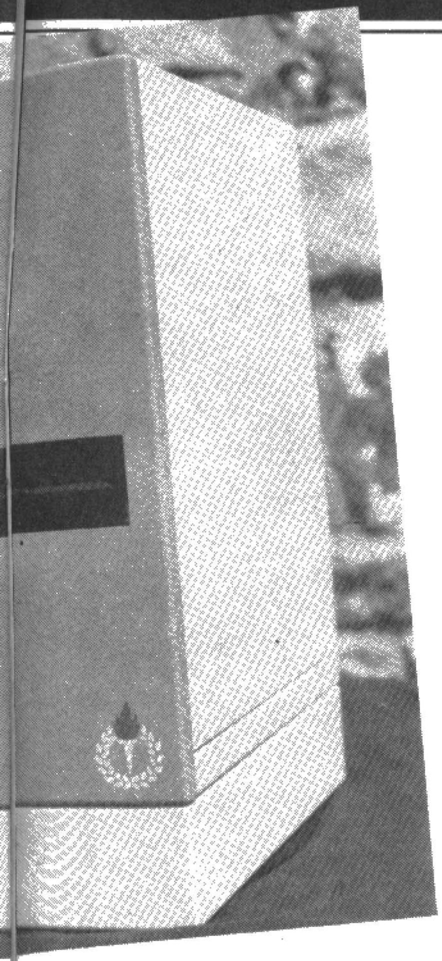
The hidden cost in the Unicorn system is, of course, the need for a BBC Micro. This not only provides a high quality 80-column VDU and colour graphics display but also handles the disc interfaces. In the case of the floppy disc the BBC Micro's standard disc controller hardware is used. This implies that not only do you have to have a model B BBC Micro but also the disc interface and filing system ROM fitted.

The only support that the Unicorn provides for the floppy disc is a power supply and a convenient cabinet – the rest is down to the BBC Micro. In the case of the hard disc things are very different and the Unicorn contains two medium sized PCBs packed with the necessary electronics to send and receive data over the 1MHz bus.

The main PCB containing the Z80 and the 68000 is not directly connected to either of the two disc drives and in this sense you can think of the Unicorn as being composed of three entirely separate subsystems:

1. a 5" floppy disc interfaced via the usual BBC Micro disc interface
2. a 5" Winchester interfaced via the 1MHz bus
3. the processor board interfaced via the tube

Indeed Torch do sell the dual disc combination in the same box for use with their Z80 second processor card; this is mounted inside the BBC Micro's case. By looking at the price for this combination – £1995 – and doing a little arithmetic it is not



difficult to work out that the cost of the Z80/68000 processor board is only £500 and the Unix software is, a little less reasonably (but outside Torch's control) £400. The only conclusion to come to is that the Winchester is still a little on the expensive side. You may be wondering if you can have the advantages of the 68000 without having to invest in a Winchester disc? The answer, but only in theory, is "no" because at the moment at least a 68000 without a Winchester disc is like a fish out of water. The main problem is that Unix needs a hard disc to operate. One possible solution might be to use the 68000 version of OS9 (see the July issue of *E&CM*) but currently Torch do not support this or any other operating system for the Unicorn apart from Unix.

Using the Unicorn

Setting up the Unicorn is fairly straightforward, involving three ribbon cables and the insertion of an extra ROM in the BBC Micro. It is fair to say that Torch normally like to do this sort of thing for the user, but after being told of the number of machines that I had pulled to bits in the course of reviews they agreed that I could attempt it! My only real criticism of the system is the short length of the ribbon cables connecting it to the BBC Micro. I appreciate the reasons for needing to keep such cables short, ie to minimise noise and hence errors, but I still found it a little frustrating to have the two units "tied" so closely together.

When the Unicorn and the BBC Micro are first switched on the Z80 system is "booted" up running CP/N – Torch's equivalent of the well known CP/M operating system. It is quite easy to change to good old BBC BASIC under software control but this looses all of the power of the Z80 and the 68000 of course. While running CP/N the floppy disc is configured as drive A: and the Winchester is divided into a number of 'logical' drives. For example drive B: and C: both refer to different storage areas of the same Winchester disc. This convention of using logical drives to divide up a large capacity Winchester is by no means unusual but it does limit the size of the largest file that you can store. In the CP/N mode the Unicorn can run any of the standard software that can be found on the other Torch computers and much of the available CP/M software. In this sense the Unicorn does turn the BBC Micro into a workable business machine but as the Z80 second processor alone does this the importance of the Unicorn is its ability to run Unix.

"Overall the quality of construction is very good".

The Unix operating system is stored on the Winchester disc in areas corresponding to logical drives D: and E: although if you are not using CP/N extensively more can be allocated. The entire Unix operating system is supplied on a large number of floppy discs and before you can run Unix the kernel and a few essential programs have to be transferred to the Winchester. Following this the operating system can be loaded simply by typing UNIX. Unicorn Unix is a standard Unisoft implementation of Unix III including the vast collection of programs that have come to be associated with it. It should be noted at this point that none of the programs are directly useful to anyone interested in tackling an application apart from text processing. Unix is essentially an environment for program development; if you want business software then you will have to buy it as an extra. (If you would like to know more about the characteristics of Unix then see the article on OS9 in July's *E&CM*.)

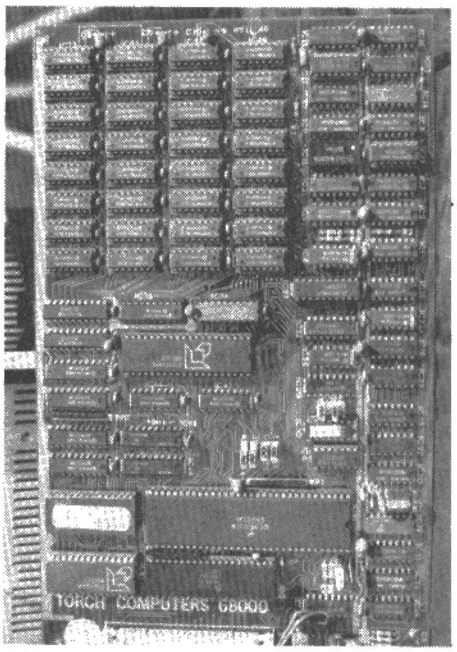
In the short time I had to test it I can only say that Unicorn Unix works as well as any other implementation that I have used. The 256K of RAM is considered to be about the smallest memory that produces a workable system and if I was going to be using Unix on a regular basis I would allocate more of the storage used by CP/N for Unix itself. The floppy disc is a very reasonable way of transferring Unix files and data to and from the hard disc and the Winchester is indeed fast.

Documentation

The only thing I can say about the documentation is that it was disappointing. Although Unix was accompanied by three huge volumes of documentation there was little of use to the complete beginner. Torch recognise this fact and do recommend a number of introductory books on the subject but this is not an approach to documentation that is to be encouraged. As far as hardware specific documentation is concerned there was virtually none! Apart from a few loose sheets indicating how to start the system and how to use the hard disc and other Torch-provided utilities there was no information on the hardware or software configuration – not even a memory map! This may be a suitable approach for the sort of user that Torch has in mind for the Unicorn but it left me feeling a little in the dark. Finally, it is worth mentioning that the three Unix manuals that come with the system really are for reference only and you do need a Unix primer to come to grips with the system.

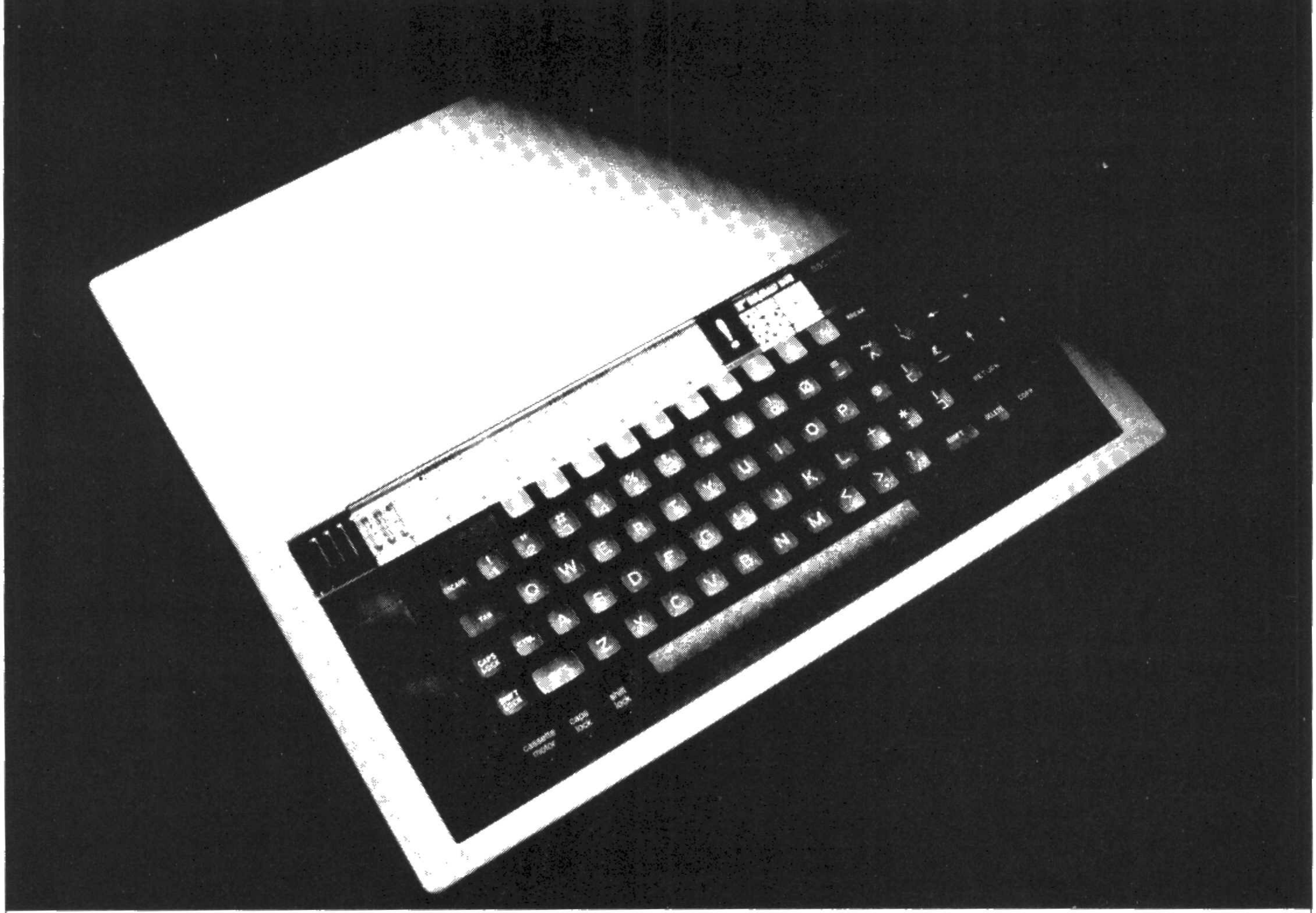
Who's Unicorn?

The Unicorn is an excellent piece of hardware and it certainly does offer a 16/32 bit second processor for the BBC Micro, at a reasonable price if you take into account the presence of the Winchester disc. However after using the system for a while I cannot help thinking that Unix still has some way to go before it reaches any sort of acceptance within the personal computer community. After spending £2895 on a Unicorn system plus the purchase of a BBC Micro, there is still not much that you can do apart from learn about Unix unless you are prepared to spend more money on applications packages. Of course, the system can be seen as a sort of insurance policy – the Z80 CP/N system for now and the Unix system just in case it does turn out to be THE operating system of the future – and for lots of users this strategy makes the Unicorn a sensible choice.



A high quality of construction is evident from this view of the main logic board.

•LOGIC STATE MONITOR•



Paul Beverley shows how the BBC micro can be used to monitor and display the logic states of a number of lines

There are a number of possible applications for this system. At its simplest, it could be used as an IC test clip to monitor the states of the 16 pins of an IC (or more than 16 as will be explained later). If certain lines are made available on the computer to provide digital outputs, we could set up a system to test automatically whether certain IC's are functioning properly. This could be extended to testing out complete digital systems, making it possible to build an automatic test station very cheaply.

Starting simply

If you only want to monitor 8 lines then you don't need to use the VIA's shift register – you can use the user port directly. A simple program for this is given as **Listing 1**, which displays the states of the eight lines as a series of 1's and 0's.

The eight lines are read into the variable A% (line 30), the cursor is moved to a particular place on the screen (40), and the FOR-NEXT loop (50 – 80) displays this number in binary. This is done repeatedly by printing the least significant bit on the screen (60) and then dividing by two in order to move all the bits down one place. The result is not a true binary equivalent of A% since it is in the wrong order, ie the least significant bit is on the left, but this should not matter as we are only interested in the states of individual lines. Thus the display gives the states of PB0 to PB7, left to right, across the screen.

LISTING 1

```
 1Ø CLS
 2Ø REPEAT
 3Ø   A%=?&FE6Ø
 4Ø   PRINT TAB(Ø,3) " ";
 5Ø   FOR N%=1TO8
 6Ø     PRINT A% AND 1;
 7Ø     A%=A%/2
 8Ø     NEXT
 9Ø   UNTILØ
1ØØ END
```

A 16 line monitor

If you want to monitor more lines than eight, say 16 or more, then one way would be to use octal latches which, with a bit of address decoding, could be put onto the 1MHz bus. However, a simpler and cheaper way is to put two or more parallel-in serial-out (PISO) shift registers together and read them into the VIA shift register serially. The circuit diagram (**Figure 1**) shows just how straightforward the circuit is: it consists of two 74LS165 PISO shift registers linked in a closed loop, clocked from CB1 and with PB0 providing a load/shift control line; the parallel inputs to the PISO registers are then linked up to the lines under test, and the data on them is latched into the registers using the PB0 line; this data is then automatically shifted into the VIA shift register, on the CB2 line, using the output from the CB1 line to clock the external shift registers.

The software needed will depend on the particular application, but three programs (**Listings 2, 3 & 4**) are given in order to show the techniques used, and to provide a means of testing out the hardware. **Listing 2** uses only BASIC, to show that it is not necessary to go into machine code provided you aren't worried about speed. **Listing 3** provides the machine code

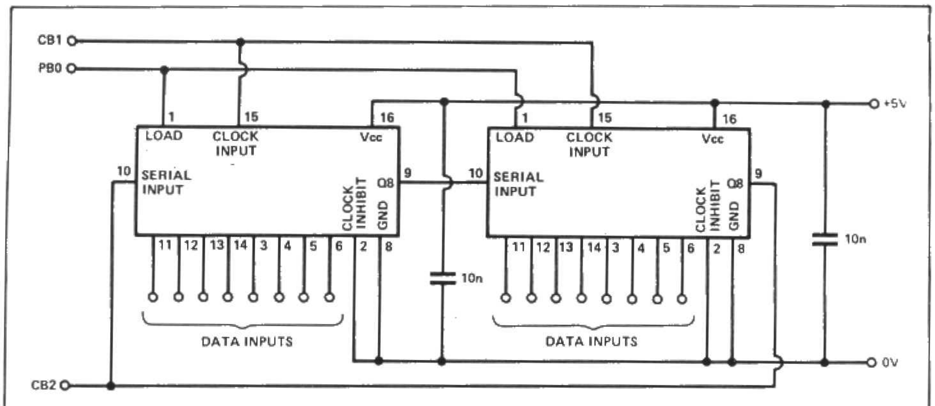


Figure 1. The circuit is based on two PISO ICs.

routines needed for high speed monitoring and **Listing 4** adds the facility for checking whether any of the lines are changing state.

One confusion tha may occur with the hardware as it stands is that – because of speed differences – the VIA shift register clocks *after* the external shift registers. The first bit is missed and the second is read in first of all. One solution is to clock the external and internal VIA's separately by generating pulses on two of the user port lines. In this way, you could ensure that the VIA was clocked *before* the external shift register.

Alternative approach

But this is a cumbersome method. A better alternative is to feed back the output of the first shift register into the input of the second, which causes the data to circulate. In this way the bit which was missed re-appears as the very last bit to be clocked into the VIA. This is the reason for the complicated-looking calculation at line 220 in **Listing 2,** which shifts and rotates the binary number back into its original position. The equivalent line in **Listing 3** (line 170) is not as complicated because the machine code program has already put the two bytes from the two registers into the single variable, A%.

However, this software correction would not be necessary in a practical system. All you have to do is to label the 16 data input lines according to the positions in which they appear in the two bytes of data acquired, regardless of where they are actually connected in the circuit. However, for the purposes of testing the system, it is easier if the displayed bits bear a direct relationship to the eight lines going into each shift register.

Program analysis

LISTING 2

After initialising one or two variables and clearing the screen, the PB0 line is set up as output by putting a 1 into the data direction register (line 40). The auxiliary control register is used to set the shift register into the "shift in under control of the system clock" mode (line 50, 010 in bits 4, 3 and 2 ie 00001000 in binary).

Lines 80 and 90 generate a pulse on PB0 which causes the shift registers to be loaded from the parallel data inputs. At line 110 there is a "dummy read" of the shift register which is needed because the clocking sequence on the VIA shift register is not initiated until you actually read the shift register (&FE6A). The number which this dummy read produces will in fact be a repeat of a previous reading and is therefore ignored. The two bytes which give the value of the data that has just been latched in, are read into variables X% and Y% (lines 120, 130).

PROCcalc takes these two bytes and makes them into a single 16-bit variable which has been rotated so that the 16 bits relate directly to the 16 input lines. It is not strictly necessary to do this rotation, but it does make it easier when setting up the system.

The display routine, PROCdisplay, is simply an extension of the routine in **Listing 1.** It displays A% as two sets of eight bits separated by a space.

LISTING 2

```
 10 ShiftReg = &FE6A
 20 PB = &FE60
 30 CLS
 40 ?&FE62 = 1 : REM Data direction Register
 50 ?&FE6B = 8 : REM Auxilliary control register
 60
 70 REPEAT
 80   ?PB = 0  : REM Load external S.R.
 90   ?PB = 1  : REM External S.R. to shift
100
110   Dummy = ?ShiftReg
120   X% = ?ShiftReg
130   Y% = ?ShiftReg
140
150   PROCcalc
160   PROCdisplay
170   UNTIL0
180 END
190
200 DEF PROCcalc
210 REM ********
220 A% = X%/2 + Y%*128 + (X% AND 1)*&8000
230 ENDPROC
240
250 DEF PROCdisplay
260 REM ***********
270 @% = 2
280 PRINT TAB(0,3) " ";
290
300 FOR N% = 1 TO 8
310   PRINT A% AND 1;
320   A% = A%/2
330   NEXT
340
350 PRINT"   ";
360
370 FOR N% = 1 TO 8
380   PRINT A% AND 1;
390   A% = A%/2
400   NEXT
410 ENDPROC
```

LISTING 3

Much of this is similar to **Listing 2,** except that the reading of the registers is done by a machine code routine called "read" (lines 440 – 640). The other difference is that this routine puts the two bytes straight into the variable A% which simplifies PROCcalc. The machine code section is as follows:

450 Put 4 in the accumulator, ready to be used in the "wait" routine.

460 – 490 produce a pulse on PB0

510 read the shift register and wait for it to complete its clocking, but ignore the result.

520 read SR again

530 put result (returned in X register) into second most significant byte of A%

540 read SR, but don't bother to wait for it to clock in since it is only the number already in the SR that we want. The number that will be clocked in as a result of having read the SR will actually be a repeat of the byte previously read.

550 put result in low byte of A%

560 end of "read" routine

590 read the shift register

620 start of wait routine – check whether bit 2 of the VIA flag register is set (accumulator already contains 4). This is the "shifting completed" flag.

630 if bit 2 is still equal to zero, go back and look again.

(If you really want to optimise the speed of this subroutine, which is only used twice, you could put this code into the main body of the program at the two places where it is needed, remembering of course to use a different label for each branch – say ".wait1" and ".wait2".)

LISTING 3

```
 10 ShiftReg = &FE6A
 20 PB = &FE60
 30 PROCassemble
 40 ?&FE62 = 1
 50 ?&FE6B = 8
 60 CLS
 70
 80 REPEAT
 90   CALL read
100   PROCcalc
110   PROCdisplay
120   UNTIL 0
130 END
140
150 DEF PROCcalc
160 REM ********
170 A% = A%/2 + (A% AND 1)*&8000
180 ENDPROC
190
200 DEF PROCdisplay
210 REM ***********
220 @% = 2
230 PRINT TAB(0,3) " ";
240
250 FOR N% = 1 TO 8
260   PRINT A% AND 1;
270   A% = A%/2
280   NEXT
290
300 PRINT"   ";
310
320 FOR N% = 1 TO 8
330   PRINT A% AND 1;
340   A% = A%/2
350   NEXT
360 ENDPROC
370
380 DEF PROCassemble
390 REM ************
400 FOR opt% = 0 TO 2 STEP 2
410   P% = &C00
420   [OPT opt%
430
440   .read
450   LDA #4
460   LDY #0
470   STY PB
480   LDY #1
490   STY PB
500
510   JSR read_wait \ dummy read
520   JSR read_wait
530   STX &405      \ A% 2nd byte
540   LDX ShiftReg
550   STX &404      \ A% low byte
560   RTS
570
580   .read_wait
590   LDX ShiftReg
600
610   .wait
620   BIT &FE6D
630   BEQ wait
640   RTS
650
660   ]
670   NEXT
680 ENDPROC
```

LISTING 4

The very useful extra facility provided by **Listing 4** is that it checks whether any of the lines are changing state. The display it produces shows, as do many of the commercial IC test clips, which of the lines are permanently high, which are permanently low, and which are continually changing, ie have some sort of square wave applied to them. This is done by using the exclusive-OR function. If you take two samples of all the data lines and exclusive-OR the two numbers together, then lines which are in the same state in both samples give a zero bit in the result, but those bits which are zero in the first sample and 1 in the second, or vice versa, will produce a logic 1 in the answer. The results of the various samples are logically OR'ed together, so that any line which has changed at any of the test times comes up with a logic 1.

Although this is actually done in machine code, we can do the same thing in BASIC. If say four test values (A1%, A2%, A3%, A4%) are obtained, then the number (B%) showing which lines have changed is given by:

B% = (A1% EOR A2%) OR
(A2% EOR A3%) OR (A3% EOR A4%)

Much of **Listing 4** is similar to **Listing 3,** but the differences are as follows:

50 – Y% is used to specify the number of times that the input lines should be read, checking each time for a change in any of the lines.

PROCcalc is omitted and it is assumed that the individual data lines would be identified by their position in the displayed value of A%.

160 The ACR is set up in such a way that PB7 is enabled to give a pulse train output which can be used to test the system.

170 The high byte of timer 1 is set to start off the PB7 pulses train.

180, 190 both A% and B% are initialised to zero in case they had been used in a previous program.

250 – 360 "PROCdisplay" is similar to **Listing 3** except that A%, which is re-named –value%" for clarity, is actually the last value that was returned by the machine code routine, and B% (re-named "changed%") is the number which has logic 1's in those bits which have changed during the course of the test.

310 this line checks it a bit has changed and if so backspaces and over-writes the 1 or 0 given by "value%" with an X to show that it is changing.

320 adds some spaces in between the displays of the two bytes.

440 the start of the machine code routine which reads the external shift registers a number of times as specified by the contents of the Y register.

450 – 470 initialise B% to zero (could be done in BASIC immediately before "CALL read" at line 60.

LISTING 4

```
 10 PROCinitialise
 20 CLS
 30
 40 REPEAT
 50   Y% = 100
 60   CALL read
 70   PROCdisplay
 80   UNTIL 0
 90 END
100
110 DEF PROCinitialise
115 REM **************
120 SR = &FE6A
130 PB = &FE60
140 PROCassemble
150 ?&FE62 = 1
160 ?&FE6B = 8 + 192 : REM 192 for PB7 pulses
170 ?&FE65 = 6 : REM only for PB7 pulses
180 A% = 0
190 B% = 0
200 ENDPROC
210
220 DEF PROCdisplay
230 REM ***********
240
250 @% = 2
260 PRINT TAB(0,3) " ";
270 value% = A%
280 changed% = B%
290 FOR N% = 1 TO 16
300   PRINT value% AND 1;
310   IF (changed% AND 1) PRINT CHR$(8);"X";
320   IF N% = 8 PRINT"    ";
330   value% = value%/2
340   changed% = changed%/2
350   NEXT
360 ENDPROC
370
380 DEF PROCassemble
390 REM ************
400 FOR opt% = 0 TO 2 STEP 2
410   P% = &C00
420   [OPT opt%
430
440   .read
445   \****
446
450   LDA #0          \ Set B% equal to zero,
460   STA &408        \ low byte and
470   STA &409        \ high byte.
480
490   .read_again
495   \**********
496
500   LDA #0          \ External SR's
510   STA PB          \ into load mode.
520   LDA #1          \ External SR's
530   STA PB          \ into shift mode.
540
550   LDA SR          \ Dummy read
560   JSR wait_shift \ Wait for 1st byte to shift in.
570   LDA SR          \ Read it.
580   TAX             \ Keep a copy of it.
590   CPY #Y%         \ Does Y contain Y% ?
600   BEQ store1      \ i.e. First time through?
610
620   EOR &405        \ Changed from last value?
630   ORA &409        \ Record any changes
640   STA &409        \ in B%.
650
660   .store1
665   \******
666
670   STX &405        \ Record latest value.
680
690   JSR wait_shift \ Wait until next byte is in.
700   LDA SR          \ Read it.
710   TAX             \ Keep a copy.
720   CPY #Y%         \ Does Y contain Y% ?
730   BEQ store2      \ i.e. First time through?
740
750   EOR &404        \ Changed from last value?
760   ORA &408        \ Record any changes
770   STA &408        \ in B%.
780
790   .store2
795   \******
796
800   STX &404         \ Record latest value.
810   DEY
820   BNE read_again
830   RTS
840
850   .wait_shift
855   \**********
856
860   LDA #4           \ Bit 2 is the SR flag.
870   .wait
875   \****
876
880   BIT &FE6D        \ Check SR flag.
890   BEQ wait         \ If not set then wait.
900   RTS
910
920   ]
930   NEXT
940 ENDPROC
```

The comments on the routines themselves should explain line by line what is happening. Basically the same testing procedure has to be done with each byte as it comes in. The X register is used as a temporary store for the latest value obtained (580 and 710), and provided this isn't the first time that the byte is being read (590, 600 and 720, 730), you check whether it is the same as the last reading (620 and 750). This gives 1's in the accumulator for each bit that has changed, and is then logical OR'ed with the B% byte to record which bits have changed at any stage in the testing (630, 640 and 760, 770). Finally, the latest value is stored in A% (670 and 800). When both bytes have been checked in this way, Y is decremented (810) and if the count has not yet reached zero (820), you go back and read it again. The "wait_shift" routine is used as before, to wait until bit 2 of the flag register is set, to indicate completion of the shifting process.

Timing problems

Finally, going back to the hardware, during development of this system a timing problem occured, similar to that which was described last month when using the shift register to establish a serial communications system. This resulted in unreliable data acquisition. However, by improving the neatness of the circuit layout, and providing adequate power supply de-coupling this problem was overcome.

If bad data acquisition occurs, then there is one very simple solution. If you put an inverter on the CB1 output line before it is applied to the clock input of the shift registers, the external shift registers are clocked on the negative edge of the pulse from CB1 instead of the positive edge on which the VIA register shifts. This allows time for the data on the CB2 line to settle before the VIA reads it. Any low power Schottky TTL gate would do, but to avoid adding a complete chip in order to provide this single inverting function, you could use a single transistor as an inverter.

The other possible solution would be to use two port lines, say PB1 and PB2, to provide separate clock pulses for the VIA and external shift registers. These pulses would have to be generated by poking the relevant numbers into the PB port, and the shift register would have to be switched into the 011 mode (shift IN under external clock). Although this is quite feasible, it is slower and, to my mind, it lacks the elegance of the solution given.

Extending the system

If you want to use more than 16 lines, then all you have to do is put more 74LS165's into the circuit, simply making them part of the closed loop so that the Q8 output of each is fed to the serial input of the next. The software can then be extended to take account of the extra bytes of information being read in.

If you need some digital outputs to set up various test conditions for an automatic test system the lines PB1 to PB7 are available, and if you want more than seven outputs then you could use the Printer Port which would provide another 8 output lines.

Feedback

If you have any further ideas for applications of the shift registers, or indeed any other part of the 6522 VIA which you think we could develop, then drop a line to the Editor and we will see what we can do. We're working at the moment on a way of speeding up the operation of the VIA, and will be looking at some more software techniques for simplifying the use of the VIA.

CBM64 FACT SHEET

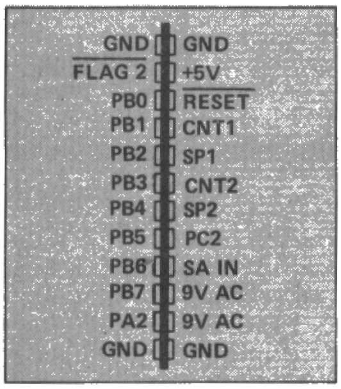
The Commodore 64, despite its games machine image, features a high technical specification. Robert Penfold presents a handy guide to the technical side of the computer.

Although the Commodore 64 has a useful range of built-in interfaces and is also one of the most popular of home computers, for some reason it has not received a great deal of attention in the technical press.

The user port

Probably the interface of most interest from the point of view of the user who wants to expand his/her machine is the user port. This is basically port B of a 6526 CIA (complex interface adaptor), which is a chip designed and manufactured by Commodore. It could reasonably be regarded as a modified 6522 VIA, a device which has received a great deal of attention in previous issues of *E&CM*.

In common with the 6522, the 6526 has two 8-bit input output ports plus handshake lines, a serial register, and two 16-bit counter timers. It also has a "time of day" clock, although this seems to be of less practical value than one might expect. The two 8-bit ports are programmed in the same way as the 6522, with a data direction register used to set each line as an input or an output. A full list of the addresses and registers for CIA 2 is provided in **Table 1**.



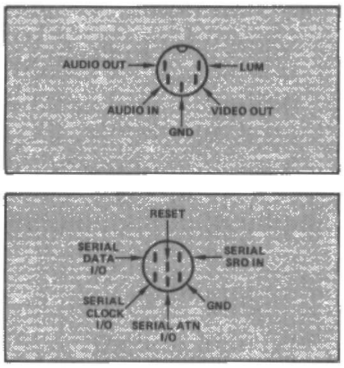
User Port.

PB0 to PB7 are the eight data lines of the port, while PC2 is a handshake output which goes low for one clock cycle following a read or write operation at port B. Flag 2 is a negative sensitive handshake input (which sets bit 4 of the interrupt control register). There is a third line that can act as a handshake (or general purpose I/O line), and this is PA2 which is merely bit 2 of port A.

The timers are similar to those of the 6522, but are perhaps a little more versatile and are programmed in a totally different way. Input pulses on CNT2 can be counted, and either a pulse of squarewave outpút can be provided on PB6/7 if the timer mode is selected. Underflows of both timers can provide interrupts. The 6526 is a very complex device, but the *Commodore 64 Programmers Reference Guide* contains a data sheet for it.

TABLE 1. Addresses and registers of the CIA 2 chip

56576	Port A
56577	Port B
56578	Data direction register A
56570	Data direction register B
56580	Timer A low byte
56581	Timer A high byte
56582	Timer B low byte
56583	Timer B high byte
56584	10ths second register
56585	Seconds register
56586	Minutes register
56587	Hours-AM/PM registers
56588	Serial register
56589	Interrupt control register
56590	Control register A
56591	Control register B

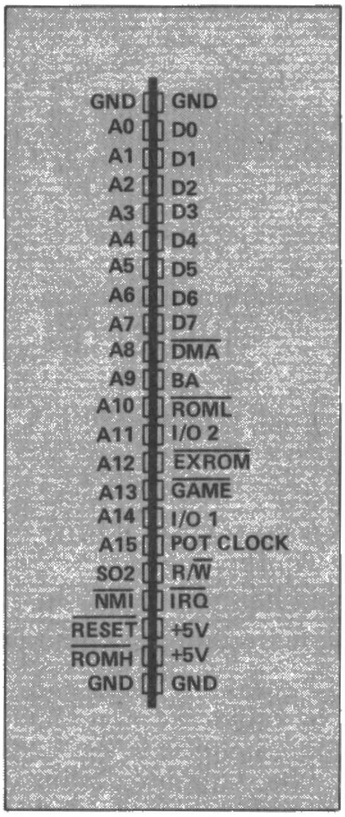


Audio/video port (above) and serial I/O (below).

Other useful lines available at this port are the main Reset line, a +5 volt output, and a 9 volt AC supply which can be used as an easy way of generating a negative supply. The maximum current drain from these two supplies is 100 milliamps and 50 milliamps respectively. Connection to the User Port is via a 2 by 12 way 0.156 inch edge connector.

The cartridge slot

If the user port does not give sufficient I/O lines for any particular application, the ROM cartridge slot can act as a general purpose expansion port which has massive I/O potential. The full data, address, and control buses are available, plus a number of other lines. Most of these are concerned with switching out the internal RAM and connecting an external ROM in its place. However, I/O 1 and I/O 2 are two decoded address bus outputs which pulse low when any address in the appropriate page is accessed (like NPCFC and NPCFD of the BBC machine's 1MHz Bus). The relevant address blocks are 56832 to 57087 for I/O 1, and 57088 to 57343 for I/O 2 (pages DE and DF respectively). If only one or two input output devices are connected to this port I/O 1 and I/O 2 can be utilised without the need for any further address decoding. By decoding these lines with the eight least significant address lines up to 512 I/O addresses are available, but bear in mind that the address, data, and control buses are all unbuffered.

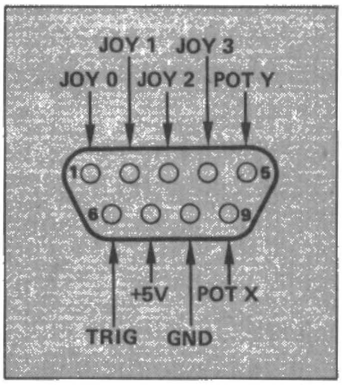


Cartridge Port.

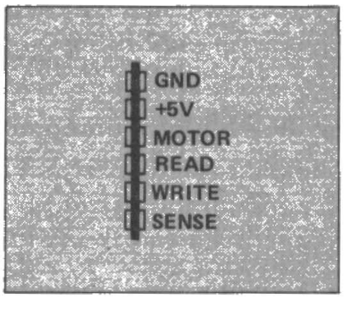
A +5 volt output is available, and the total current drain on this and the user port's +5 volt output must not exceed 450 milliamps. A clock output is provided, and this has a nominal frequency of 980kHz. All system timing is derived from the 8.18MHz "dot clock", and an output at this frequency is also available. Connection to the cartridge slot is by way of a 2 by 22 way 0.1 inch pitch (male) edge connector.

The cassette port

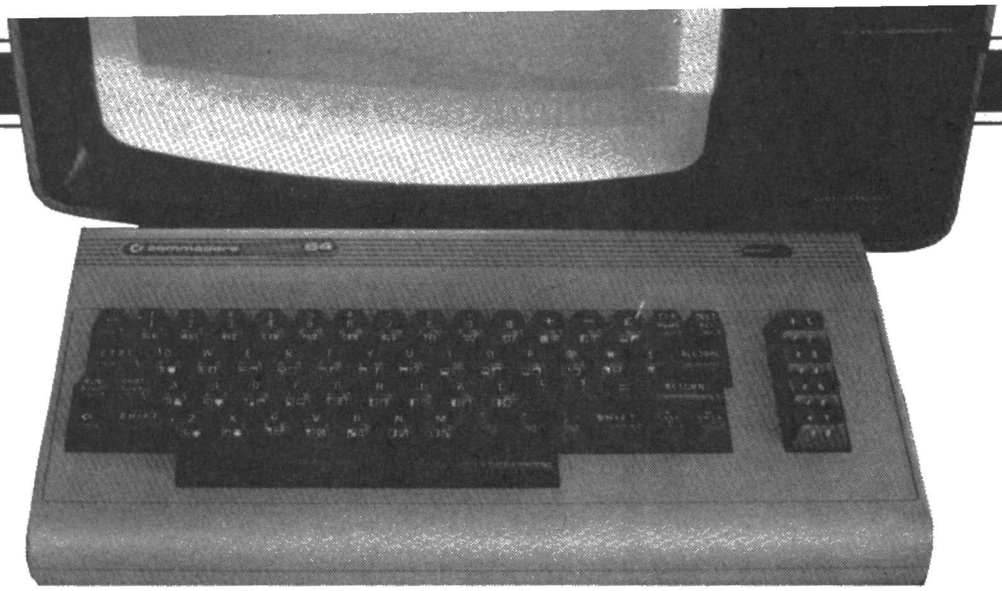
The Commodore 64 has a dedicated cassette recorder, and it does not normally seem to be possible to interface an ordinary recorder direct to this port. The MOTOR output provides a nominal 9 volt supply which directly provides power for the motor in the recorder, and also gives automatic start/stop control. READ and WRITE are the data input and output terminals respectively. The output is at standard 5 volt logic levels, and the computer is designed to receive the same type of signal. The SENSE terminal is an input which is taken low when the play key of the recorder is activated. This port could be connected to a normal cassette recorder if the



Game Port.



Cassette Port.



SENSE terminal was connected to earth, the (port) output signal suitably attenuated, and a signal conditioning circuit used to provide the READ input with standard logic levels. The MOTOR output could be used to operate a relay and control the cassette motor via the REM socket. The cassette port uses a 2 by 6-way 0.156 inch pitch edge connector. Note that the lower set of terminals merely duplicate the upper set.

Games ports

There are two standard Atari/Commodore style joystick ports on the Commodore 64, and there are numerous switch type joysticks which connect to these 9-way D plugs. It is not advisable to use these inputs for other purposes since they are shared with the keyboard. There are two analogue inputs on each games port (Pot X and Pot Y), but these inputs are just duplicated on the two ports and there are actually only two inputs available. These inputs are not the normal voltage sensitive types. Instead they are intended to directly sense the resistance of the potentiometer in a games paddle. This resistance must be connected between the input and the +5 volt supply, with a value of about 400k giving a full scale reading. The full scale resistance can be reduced by connecting a capacitor from the relevant input to the negative supply rail. The converters can be read at addresses 54297 and 54298.

Other ins and outs

The audio/video port can be used with a Commodore monitor or any monochrome monitor having a standard composite input. Audio input and output terminals are also provided. The serial input/output port enables Commodore peripherals such as the 1541 disc drive to be connected to the machine. The system used here is sort of serial version of the IEEE-488 parallel interface, and an adaptor is needed in order to use normal IEEE-488 equipment with this port.

TABLE 2. Primary system variables of the CBM64

Dec. Address	Hex. Address	Contents
45–46	2D–2E	Start of BASIC variables
47–48	2F–30	Start of BASIC arrays
49–50	31–32	End of BASIC arrays +1
51–52	33–34	Bottom of string storage (BASIC)
55–56	37–38	Highest address used by BASIC
147	93	Cassette flag (0 = load, 1 = verify)
154	9A	Default output (CMD) device
160–162	A0–A2	3 byte 1/60th second timer
197	C5	Current key pressed (0 = no key)
211	D3	Cursor column on current line
641–642	281–282	MEMSTR (Bottom of memory for O.S.)
643–644	283–284	MEMSIZ (Top of memory for O.S.)
648	288	Top of screen memory MSB (PAGE)
651	28B	Key repeat speed
652	28C	Key repeat delay

Sideways RAM revisited

Brian Alderwick and Peter Simpson

In the December 1983 and January 1984 issues of *E&CM* we described a 'sideways' RAM board for the BBC micro. This provided 16K of RAM in a position in the memory map normally occupied by a 'sideways' ROM and allowed images of ROMs to be stored on disc or cassette and used without the need to add a ROM extension board. The RAM board also allowed easy development of original software; this only needed to be put into an EPROM once fully de-bugged and working to the programmers satisfaction. The board was designed so that writing to the RAM was only possible when the RAM was selected by the 'sideways' ROM paging register; the advantage of this method was that, once loaded, the RAM was protected from any further WRITEing unless the RAM board was selected. The disadvantage was that loading and saving of the RAM board code was accomplished through additional code which needed to be loaded every time the computer was switched on.

We had requests for a hardware modification to enable direct loading of the 'sideways' RAM without the use of software. The authors also had a need for this type of operation for a 16K printer buffer which uses this RAM board when it is not in use as a 'sideways' ROM emulator.

Theory of operation

The extra circuitry provided does two things to enable the RAM to be written to directly. Firstly, the RAM board chip select line is intercepted to enable selection of the RAM whenever a write to the sideways ROM area (&8000 – &BFFF) is detected, regardless of the paging register value, as well as when a normal READ/WRITE is performed with the RAM selected. Secondly, the remainder of the ROM sockets are disabled whilst any WRITE operation to the 'sideways' ROM area is current, to prevent bus clashes.

Hardware modifications

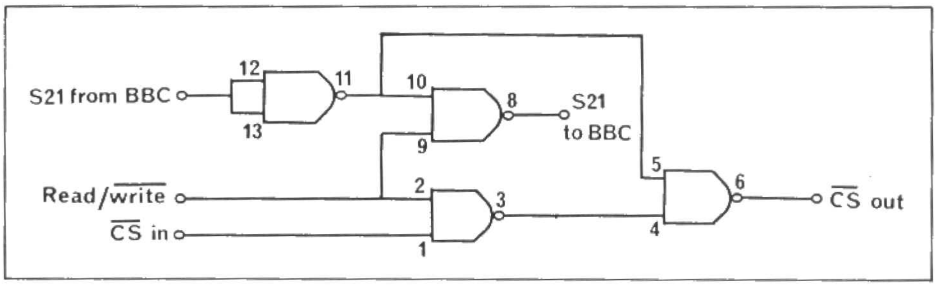

S21 from BBC
12
13
11
10
9
8
S21
to BBC
5
6
$\overline{\text{CS}}$ out
Read/$\overline{\text{write}}$
$\overline{\text{CS}}$ in
2
3
1
4

Figure 1. A single NAND gate is the only hardware addition.

All the necessary logic changes can be accomplished with one 74LS00 quadruple NAND gate, the circuit diagram is shown in **Figure 1.** The sideways ROM select signal, the RAM board select signal and the read/write line are gated together to produce a new RAM board select and a return signal for the ROM select on the BBC main board.

The prototype RAM board was modified by glueing the chip on the board with epoxy resin. It was mounted upside down and the wiring directly soldered to the pins. **Figure 2** shows the physical layout and connections.

Only one existing connection needs changing on the RAM board, this is the chip select which comes in on pin 20 of the 28-way cable. Remove this wire from the hole and solder it to pin 1 of the IC and take a new wire from pin 6 of the IC back to the vacated hole. Two new connections need to come from the BBC main board at link S21. This link is located near the keyboard connector on the main board and is the closest to the keyboard of the pair marked S21, see **Figure 3.** Remove the existing shorting link and place it on a spare pin for possible future use. Make up two wires with crimp-on connectors suitable to fit on the exposed pins of S21 south. The right hand pin (a), should connect to pin 12 of the IC and the left hand pin (b), to pin 8 of the IC. Three additional connections need to be made from the RAM board to the IC, these are, READ/WRITE to pin 9, five volts to pin 14 and ground to pin 7. Finally, interconnections on the IC itself should be made between the following points: pin 2 to pin 9, pin 3 to pin 4, pin 5 to pin 10, pin 10 to pin 11 and pin 12 to pin 13.

Readers who have built the RAM card and who experience occassional corruption should note that the most reliable place to pick up the READ/WRITE line on the main board is at IC77 pin 8 and not IC72 pin 1 as stated in the original article. This change is required because the new READ/WRITE line is clocked by the phi 2 processor clock and does not stay low into the next machine cycle which occurs with the ungated line and which can cause problems with fast RAM chips.

This completes the necessary hardware modifications and a functional test may be given by loading a known working sideways ROM image with the *LOAD progname 8000 command and then pressing BREAK to initialise it.

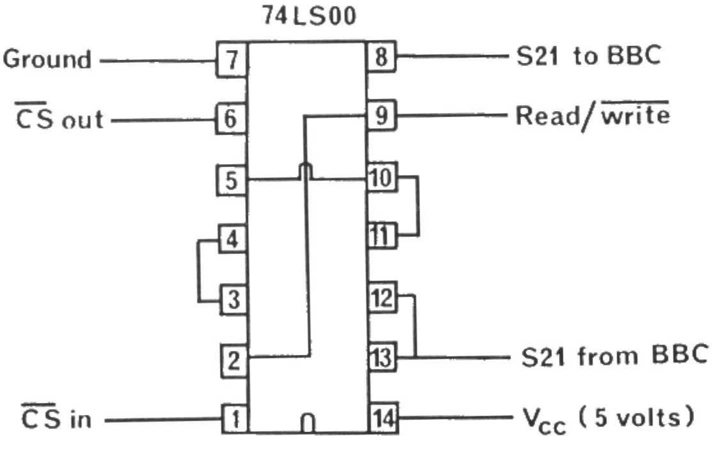

74LS00
Ground
$\overline{\text{CS}}$ out
$\overline{\text{CS}}$ in
7
6
5
4
3
2
1
8
9
10
11
12
13
14
S21 to BBC
Read/$\overline{\text{write}}$
S21 from BBC
V_{cc} (5 volts)

Figure 2. The circuit connections are made directly to the pins of the IC.

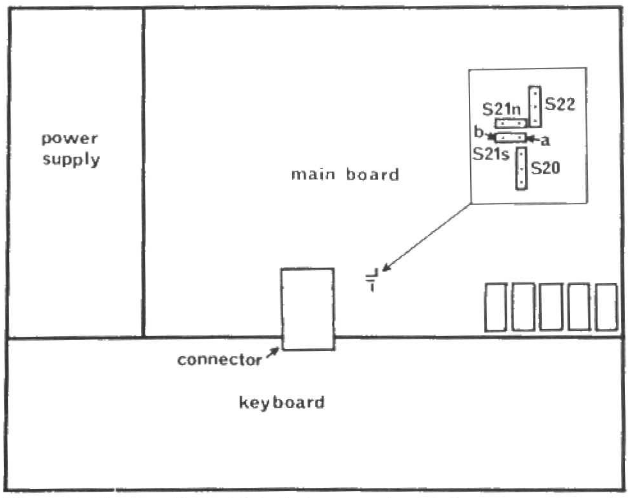


Figure 3. The location of link S21 within the BBC micro.

FREE EVERY MONTH

NOVEMBER 1984

your ROBOT

BRITAIN'S FIRST ROBOTICS MAGAZINE

Each month, Your Robot reports on the latest news from the world of low cost robotics. In addition in-depth features explain the theory behind the operation of robots and computer control systems.

NEW DAWN DEW FOR MOVITS

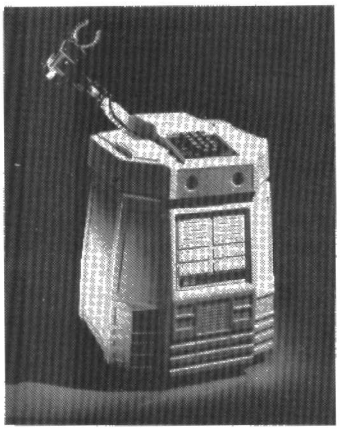
The Movit range of robot kits suffer from one major drawback in that none of the members of the family make any provision for computer control. While some of the robots are designed to react to external stimulae, sound and light (and could thus be correctly described as simple forms of robot) the fact that computer control is ruled out limits their appeal to the robotics experimenter. Dew Electronics have recognised this problem and have designed an interface that replaces the electronics of the Line Tracer Movit with a small board that may be connected to the I/O port of most microcomputers.

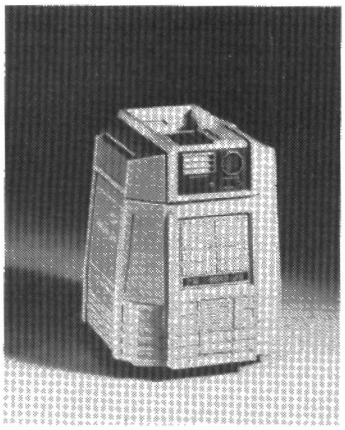
The interface adopts a different approach to the majority of robot/computer links in that it does not use a parallel connection between the two pieces of equipment, but instead features a bi-directional serial link (two wire plus screen). Next month, we shall review the interface in full, but for now we'll just say that despite the fact that the link is made via only three wires, it is possible to fully control both the Movit's motors and provide seven channels of feedback from the robot. The documentation that accompanies the interface even suggests how speed control may be implemented by a software orientated pulse width modulation system.

The fact that there are only two channels of communication between robot and computer also suggests that it would be a simple matter to implement some form of remote control link by either an infra red or radio system.

At present the interface is supplied with software written for the BBC micro but if sufficient interest is shown, and surely there will be, Dew will consider providing software drivers for other computers, notably the Spectrum.

If you do not want to wait for next month's review before buying the interface, Dew Electronics are at 21 Warwick Street, Oxford, OX4 1SZ and are retailing the board at £25 fully inclusive.





Little brother for HERO

The Heath/Zenith HERO robot now has a little brother in the form of HERO Jnr. The robot was recently launched in the States and is best described as a stripped down version of the original design. HERO Jnr features a 32K ROM in which resides what could be termed a robot operating system.

The robot has three drive wheels including a single articulated rear drive wheel that allows the robot to successfully negotiate obstacles on a variety of different surfaces.

The robot's head is home for a 17 key keypad which allows the user to modify various aspects of HERO Jnr's operation.

Also included in the robot's specification is provision for an infra-red motion detector and simple light and sound sensing systems.

We shall be returning for a more detailed look at Jnr's specifications in a future issue of **Your Robot.** *From what we've seen of this design though, it offers an attractive specification at a price, at least in the US, as quite affordable.*

Spectrum Roboticists' Aid

The name Tony Berk will be familiar to anyone who has followed the electronic press over the years. Dr. Berk, to give him his more formal title, has recently turned his attention to the world of robotics and in particular to how owners of the Spectrum computer can explore various aspects of the field.

After an introduction that sets the scene for those new to concepts of robotics, the book moves on to give practical examples of using the Spectrum in control applications. The book deals both with the way in which the computer is used to control the movement of robot arms (via relays, stepper motors etc) and the way in which sensory information (light, sound, position) is input to the Spectrum. The manner in which sensory information is used to modify the action of a robotic system is also discussed.

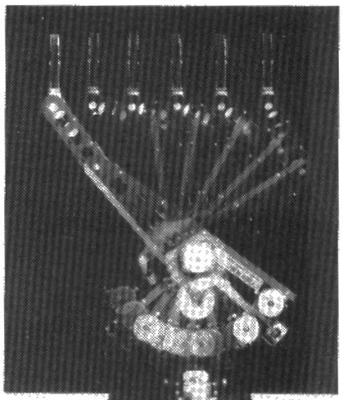
The book lists a number of commercial designs and shows how the Spectrum may be interfaced to them.

Anyone with a Spectrum and an interest in robotics will find the book invaluable. In addition, much of the information will also be of use to owners of other computers.

Practical robotics and interfacing for the Spectrum.
A. A. Berk.
Granada Publishing
£6.95
Publication date October 25th 1984.

'Your Robot' News Desk is on 01-833 0846

New Cyber software



Cyber Robotics have developed a new suite of applications software designed to widen the appeal of their Cyber 310 robot arm. The new software is designed to overcome the problems associated with the positioning of an arm using the system adopted by the majority of educational robots, that is by specifying the joint to be moved and how far it should move. With a three, or more axis robot, simple positioning of an arm can require an inordinate amount of programming.

Cyber's applications software overcomes these problems by allowing the position of the arm to be specified in either Cartesian or Polar co-ordinates. Cartesian co-ordinates are preferred when the arm is required to move to a number of points in a straight line while Polar co-ordinates are best suited to cases in which the arm is to move in a circle or to points within the same plane.

The new suite of software means that the 310 is able more closely to emulate the performance of full scale industrial robots. It also widens the educational use of the arm to include, for example, such disciplines as mathematics.

More details from Cyber Robotics at 1 Ditton Walk, Cambridge, CB5 8QZ.

NEWS

Your Robot, 155 Farringdon Road, London EC1R 3AD. **Editor:** Gary Evans, **Deputy Editor:** William Owen, **Production Editor:** Liz Gregory, **Advertisement Manager:** Richard Jansz, **Advertisement Production:** Yvonne Moyser.

BUGGY BUILDING

Last month Richard Sargent brought you Laurel, this month (surprise surprise) we have Hardy, a low cost buggy with a unique means of locomotion.

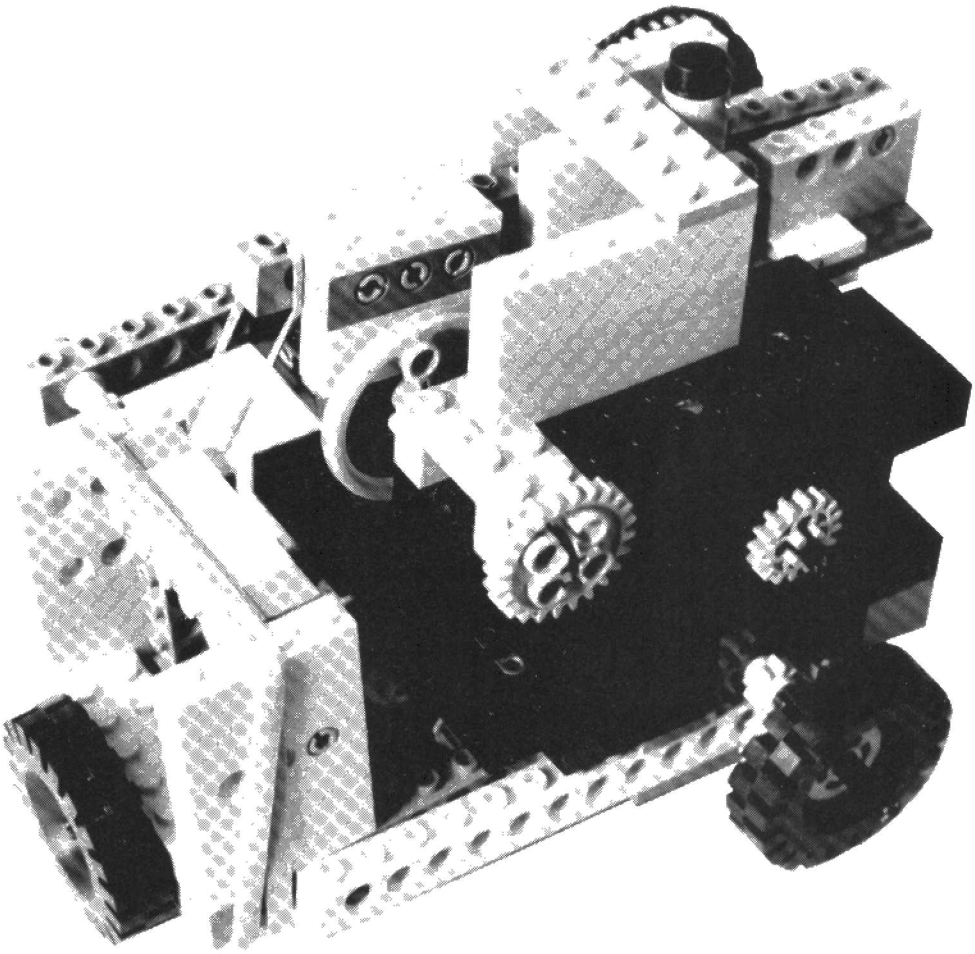
We return to the Lego construction medium with a vengeance with the introduction of Hardy, a robotic buggy, which will learn to move this month and to avoid obstacles next month.

If you have been following the Lego Robot series, you will already be familiar with the formula for our experimental constructions. Take one 8-bit Centronics port, one motor interface pcb (which drives two motors), an assortment of 4.5V Lego motors and technical beams and use standard BASIC to drive the beast.

Hardy puts the Centronics-specification under strain, but it is still possible to proceed with the original circuitry providing your computer has 8 bits of parallel output and 1 bit of input. However, for 380Z users (ie half the school population!), Amstrad CPC464 owners, and anybody else out there who has a computer with a 7-bit Centronics port – fear not! A 7-bit Centronics expansion circuit giving 11 or 14 lines of output and 16 lines of input has been designed. As your robotic designs become more complicated, you will find that a dozen input and output lines represents the minimum I/O capability required.

TURTLES

Floor crawling robots are generally known as turtles when their function is to demonstrate fundamental geometry and logical thinking via the language LOGO. Hardy could be used as a turtle – provision has been made for a retractible pen – but it is envisaged as a much more general purpose crawler, very much in the mould of the "BBC Buggy" which has now made numerous appearances on the BBC Computer TV series. Hardy is somewhat ponderous, but it is a versatile machine, capable of undergoing revision and amendments to its superstructure. By special design it is able to run effectively upon thick-pile carpets, so it is an ideal robot for the home as well as for the school . . . Could this be another first in British Robotic History?



A HARDY DESIGN

As can be seen in the accompanying sketches and photographs the base of the vehicle consists of two Lego 4.5V motors mounted side by side on a chassis of Lego technical beams. The motors are the geared type that you find in the Lego box 107. Each motor directly drives a large Lego wheel in traditional turtle style, but the "third wheel", which is usually a swivel castor or a ball-bearing castor, is a departure in turtle style. There are *two* "third wheels" and they are mounted in a frame of Technical Lego which swivels so that one or the other wheel is in contact with the ground. One wheel enables the buggy to travel in a near-perfect straight line, while the other is brought into action when the buggy needs to turn around its centre point, which is the pen position. This is the arrangement which allows Hardy to travel over carpeted surfaces which would hinder or stall a buggy with a stub for its third wheel. If you wish to follow the precise design of Hardy's chassis, detailed plans are available (see the components list). The principle of operation is shown in **Figure 1.**

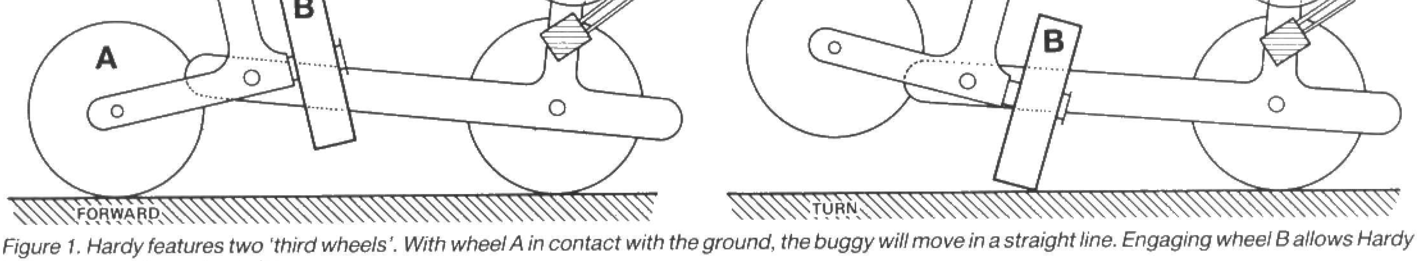


Figure 1. Hardy features two 'third wheels'. With wheel A in contact with the ground, the buggy will move in a straight line. Engaging wheel B allows Hardy to turn around its centre point.

THE SUPERSTRUCTURE

This, generally speaking, is everything which rests on top of the two drive-motors and the construction is not as critical as that required for the lower half. One motor is needed to operate the third-wheel frame, and another is needed to raise and lower the pen. The prototype uses two more "107 type" motors, but it should be equally possible to use the smaller Lego motors with the 20:1 reduction gearing (sets 8700 and 872).

Gear and chain linkages take the power from the upper two motors to suitable points on the superstructure where the rotary motion can be translated into the horizontal movements required to move the wheel-frame and the pen-slider. **Figure 2** shows the pen-sliding mechanism. The slider is a Lego axle moving freely through two Lego holed plates. These plates have been cut to allow the pen freedom of movement, which is important! Generally speaking, cutting and glueing your Lego should be a last-resort measure, since the pieces affected can't always be used again. In this case it's unavoidable. Make clean cuts with a Stanley-knife. The pen is attached to the axle by two rubber bands, and these should be positioned where they don't foul the axle supports. The motor-driven wheel has a peg in it which moves the 15-hole technical beam up and down against the resistance of a rubber band. By driving the peg-wheel constantly it is possible to make Hardy draw dotted and dashed lines (dot-matrix print-outs haven't been tried yet!) but this is not catered for in the software example shown.

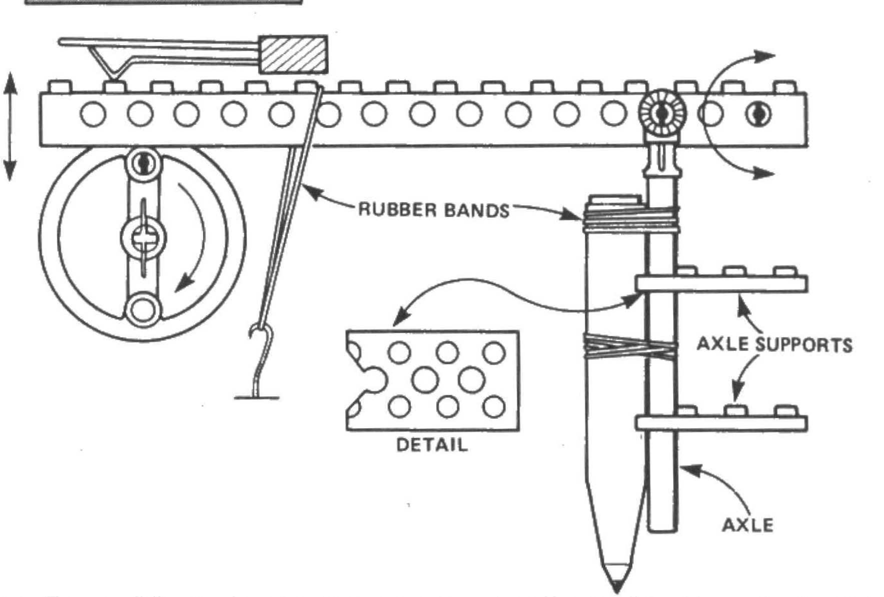


Figure 2. The pen sliding mechanism makes use of a number of Lego building blocks plus the occasional rubber band. Cutting and glueing Lego, while usually a last resort, is unavoidable in this case.

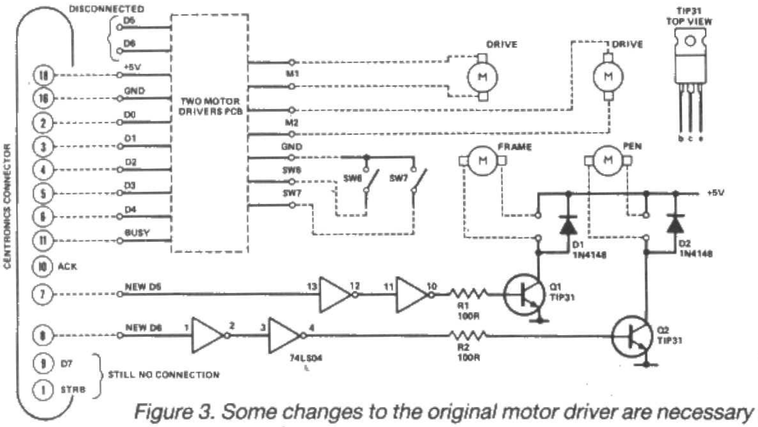


Figure 3. Some changes to the original motor driver are necessary for Hardy to operate correctly.

Notice that both the wheel-frame and the pen-slider need a micro-switch to provide feed back information to the computer. The 8-bit Centronics interface normally provides for up to 8 sensors (see page 55, *E&CM* August 1984), but with the addition of the two extra motors bits D5 and D6 have been "stolen" for drive duties, leaving D4 to read one of two sensors. These are allocated as follows:

S6	sense wheel-frame position
S7	sense pen position
S0-S5	are unavailable

This arrangement is necessary so that the circuit will work from a 7-bit Centronics port. 8-bit Centronics users can obtain a total of four inputs if they wish, by bringing the Centronics D7 signal over to point D5 on the pcb. This extends the range of the input multiplexer and allows S4 and S5 to be used as front and rear bumper contacts. However, as we're not concerned with bump-contacts this month, we will concentrate on the minimum configuration only.

CIRCUIT ONE

Figure 3 shows the additions and changes that need to be made to the original "Two Motor-Drivers" circuit of August 84. Track-cutting on the original pcb is not necessary, since if the PCB's D5 and D6 are left unconnected with 2K2 resistors R9 and R10 will take pins 9 and 10 of IC3 high and sense lines S6-S7 will become the only valid inputs, selected by the address given by D4 on pin 11. The circuit for the extra motors can be built up on vero-board. These motors always turn in the same direction and are therefore driven by a single-transistor, rather than the four needed for bi-directional operation.

The control words which may now be sent to the Centronics port are as follows:

Shut down robot	00000000	0
Left drive forward	00000001	1
Left drive reverse	00000011	3
Right drive forward	00000100	4
Right drive reverse	00001100	12
Move wheel-frame	00100000	32
Move pen	01010000	80

At first glance it might be tempting to drive Hardy jn the same way that we drove Laurel last month – by using LPRINT commands to put the data out on the Centronics lines. This worked with Laurel because the Centronics- Busy line could be held at a permanent low level by typing sensor SO to ground. If BUSY is seen as a high by the LPRINT command, the computer will wait forever in a loop until BUSY does go low. In the case of Hardy, BUSY reflects the state of a chosen sensor, and it is therefore not possible to use LPRINT, and POKE or OUT will have to be used instead.

There is, it seems, a little game that some micro manufacturers like to play with the new owners of their computers. They like to hide the single reference to the Centronics port address somewhere in their 500 page manual. I'm convinced they don't like the printer port being used for anything other than a printer! **Table 1** shows the addresses for some current (and not-so-current) Z80 micros.

This table makes the situation look deceptively simple. It isn't. Older 380Zs had their port at OFFF. The Nascom PIO has to be configured by the user. So it

TABLE 1. Z80 addresses.

	Data (write)	Busy (read)	
380Z	POKE FBFF	PEEK FBFF	(bit 0)
NASCOM (PIO)	OUT 5	INP 4	(bit x)
SPECTRUM (PIO)	OUT 65247	IN 64735	(bit 1)
MTX500/512	OUT 4	INP 4	(bit 0)
CPC464	CALL &BD31	CALL &BD2E	(carry flag)

could become INP 5 and OUT 4. The Spectrum has no ports of its own – the address given is for the PIO add-on shown on page 54 of *E&CM* August 1984. The Memotech MTX has an unusual Centronics STROBE arrangement and the Amstrad CPC464 port can only be reached by two machine-code CALLs.

There is no easy solution to these convolutions – if you want to experiment with robotics, then you must read your computer manual very carefully and find out what signals the various ports are capable of handling.

HARDY'S SOFTWARE

As the complexity of each model robot increases the pressure to turn away from slow Basic to faster Forth (which not all users will possess) and to Machine Language (different codes for different CPUs) will become ever stronger. However, the whole point of the Lego Robotics series is that the models and ideas outlined should run on as many micros as possible, and that potential builders of interesting hardware should not be discouraged by over-complex software. Therefore it is important that all the Robotic Things should at least do something under Basic, even though they may perform better when controlled by a different language.

This month's demonstration BASIC really just involves incorporating the sensor checking code that we used in the Wall-Building Robot (August) into the chain code control program that was used to drive Laurel last month.

Observant code-breakers will have noticed that the S-Stop command in the Laurel listing was in fact redundant, since all the other commands always neatly turned off the motors at the end of their time-period! We can use this fact to advantage for Hardy by using the S command for Select Pen and the code at line 700, which was the stop-code, now becomes the Pen-select code. Pen-select will act as a "toggle". Use it once and the pen will go down, use it again and the pen will rise. Note too that this month's code can't use LPRINT. The port addresses are contained in the two variables, PB (read the Busy) and PD (port data). They must be set early in the program to the correct values for your computer. The MTX is being used as the example machine so the code required is LET PB=4 and INP (PB) and OUT PD. Other machines may use PEEK (PB) and POKE PD.

Sending 7 or 8 bits to the Centronics port using OUT or POKE is no real problem but reading the BUSY line is tricky in BASIC, since it is the state of *one bit* that we're interested in, and the other 7 bits tend to get in the way! Since this is an issue of some importance, its discussion cannot be delayed any longer.

READING BITS

In machine code, the state of a single bit can often be read directly. To read a BUSY signal coming in on Bit 5 in Z80 code we would use BIT 5,A : JP Z 1000 which really means "test the state of bit 5 in register A and if it is a zero branch to some code at 1000".

Another way of testing bit 5 is to set up a mask to "blank off" the bits that are irrelevant and then test the *whole* byte. This is commonly done at machine code level, and can also be used by some BASICs. Here's how it works:

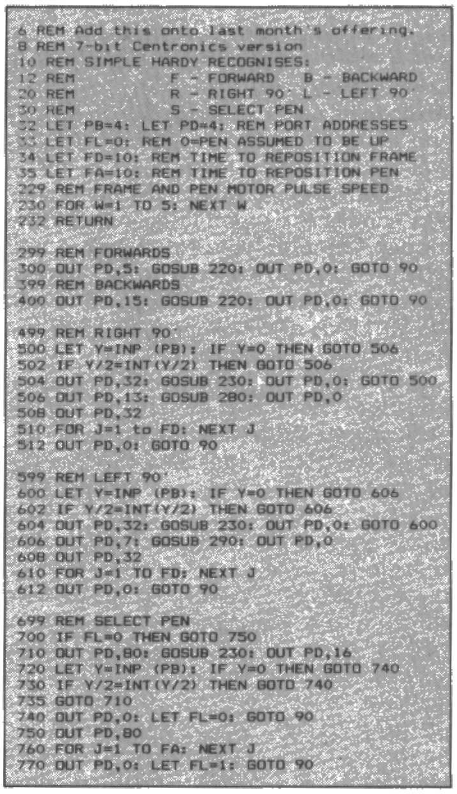
```
              B
           76543210
1 Read the port 10101010 – AAhex (170)
2 Set up mask – 00100000 – 20hex (32)
3 Perform Logical AND thus : AA AND 20
4 Result is 00100000 30hex (32)
```

Logical AND will only produce a 1 if both "inputs" are 1. In the example, BUSY bit 5 was high and the mask (32) produced the result 32. If BUSY had been low, the same mask would have produced the result 0. No other results can occur, and a test for 32 (or indeed for "non-zero") would have established the condition of the BUSY line. Clever BASICs, like that used on the Nascom, can perform this "bitwise disjunction", but do not assume that if your BASIC has the keywords AND, OR and NOT it can necessarily do bit manipulation of this sort. These Logical Operators tend to perform Boolean algebraic functions – bitwise manipulation is an added bonus. If the answer to PRINT 5 OR 6 is 7 then your machine can perform bitwise Logical Operations. In like manner PRINT 255 AND 1 will respond with 1. If you use a Spectrum then you can give your computer bitwise logic, if you haven't already done so, by referring to the May issue of *E&CM*. That will help interpret the BUSY signal which comes in rather awkwardly on Bit 1. The mask needed to isolate bit 1 is 00000010 which is 2. Fortunately most BUSY signals come in on bit 0 which is reasonably easy to detect using "ordinary" BASIC. A zero will always produce an even byte, a 1 will produce an odd byte. so, in MTX BASIC, we have:

```
LET Y=INP(4)
      : REM Read BUSY line
IF Y=0 THEN GOTO xxxx
      : REM Jump if BUSY low
IF Y/2=INT(Y/2) THEN GOTO xxxx
      : REM Equates only if Y even
        and if so BUSY low so jump
Else code continues as BUSY was high
```

The "forwards" and "backwards" code (lines 299-400) is the same as that used for LAUREL, except that OUT replaces LPRINT. The "turning" code is somewhat different.

Turn right 90° lines(499-512), and the similar turn left 90° (lines 599-612) work like this: Lines 500-504 pulses the wheel-frame motor in small units of time, given by the delay at line 230, until the sense switch S6 returns zero and causes a branch to line 506. This has the effect of placing the turning wheel on the floor. Then line 506 causes a 90° pivot, the accuracy of which is determined by the value of RD, set in line 40 and using the subroutine at line 280. Finally, the code in line 508-510 moves the wheel-frame by the time value FD. FD, in line 34, should be given a value which will bring the normal third-wheel down onto the floor.



```
6 REM Add this onto last month's offering.
8 REM 7-bit Centronics version
10 REM SIMPLE HARDY RECOGNISES:
12 REM            F - FORWARD    B - BACKWARD
20 REM            R - RIGHT 90  L - LEFT 90
30 REM            S - SELECT PEN
32 LET PB=4: LET PD=4: REM PORT ADDRESSES
33 LET FL=0: REM 0=PEN ASSUMED TO BE UP
34 LET FD=10: REM TIME TO REPOSITION FRAME
35 LET FA=10: REM TIME TO REPOSITION PEN
229 REM FRAME AND PEN MOTOR PULSE SPEED
230 FOR W=1 TO 5: NEXT W
232 RETURN

299 REM FORWARDS
300 OUT PD,5: GOSUB 220: OUT PD,0: GOTO 90
399 REM BACKWARDS
400 OUT PD,15: GOSUB 220: OUT PD,0: GOTO 90

499 REM RIGHT 90
500 LET Y=INP (PB): IF Y=0 THEN GOTO 506
502 IF Y/2=INT(Y/2) THEN GOTO 506
504 OUT PD,32: GOSUB 230: OUT PD,0: GOTO 500
506 OUT PD,13: GOSUB 280: OUT PD,0
508 OUT PD,32
510 FOR J=1 to FD: NEXT J
512 OUT PD,0: GOTO 90

599 REM LEFT 90
600 LET Y=INP (PB): IF Y=0 THEN GOTO 606
602 IF Y/2=INT(Y/2) THEN GOTO 606
604 OUT PD,32: GOSUB 230: OUT PD,0: GOTO 600
606 OUT PD,7: GOSUB 290: OUT PD,0
608 OUT PD,32
610 FOR J=1 TO FD: NEXT J
612 OUT PD,0: GOTO 90

699 REM SELECT PEN
700 IF FL=0 THEN GOTO 750
710 OUT PD,80: GOSUB 230: OUT PD,16
720 LET Y=INP (PB): IF Y=0 THEN GOTO 740
730 IF Y/2=INT(Y/2) THEN GOTO 740
735 GOTO 710
740 OUT PD,0: LET FL=0: GOTO 90
750 OUT PD,80
760 FOR J=1 TO FA: NEXT J
770 OUT PD,0: LET FL=1: GOTO 90
```

The Select Pen routine at 699 contains both the "pen up" and "pen down" routines. If FL=0 (line 700), the pen is assumed to be UP and the routine at line 750 uses the time value FA to drive the motor and so lower the pen, setting the flag FL to indicate "pen down". When a subsequent request is made to the SELECT PEN routine, FL=1 and the code in lines 710-740 is run. Note that the values 80 and 16 in line 710 control the motor whilst also selecting sensor S7. The motor is run until S7 returns zero. Line 740 reselects sensor S6, resets the FL flag and exits.

Next month Hardy acquires sensors, an extra interface board, and some Z80 machine code driving and sensing routines.

PARTS LIST

IC1	74LS04 HEX INVERTER
TR1,2	TIP 31
R1,2	100R 0.25W
D1,2	1N414B

Small piece Vero board; Copper strip for home-made micro-switches.

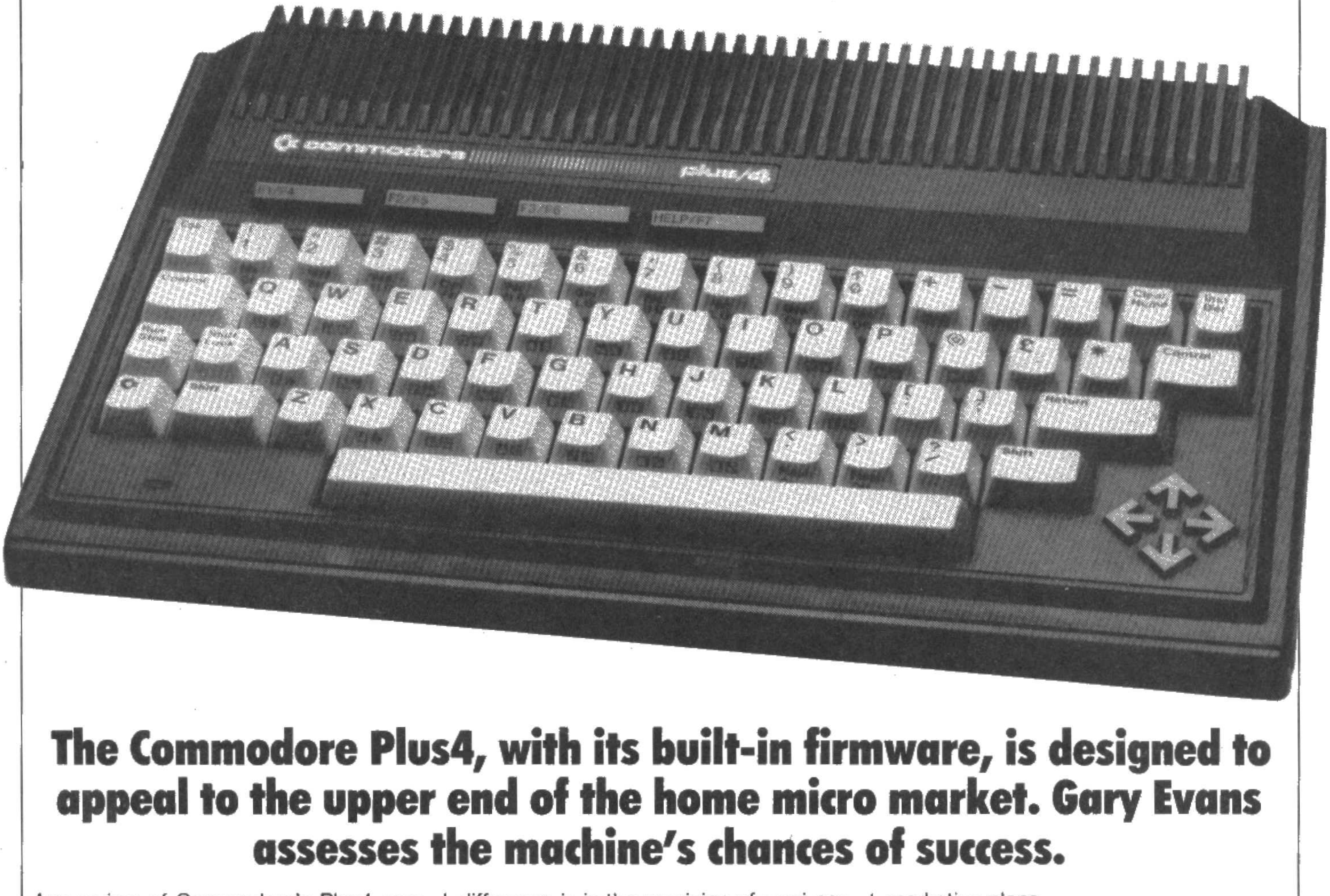
LEGO

In addition to the basic motors, builders will now find the following kits worth investigating:

8710	assorted technical pieces
872	20:1 reduction gearing & chain
1217	technical plates, short beams
1221	long beams
1226	2 medium tyres
1227	extra gears

To obtain drawing plans and a complete parts lits of the Hardy buggy, send a large stamped self-addressed envelope to *E&CM.*

PLUS 4 REVIEW



The Commodore Plus4, with its built-in firmware, is designed to appeal to the upper end of the home micro market. Gary Evans assesses the machine's chances of success.

Any review of Commodore's Plus4 computer must, as its central theme, consider the quality of the four inbuilt applications packages from which the computer derives its name. While the hardware configuration of the Plus4 features several interesting aspects, the success of the machine will depend on whether or not Commodore can convince the buying public that the inbuilt word processor, spread sheet, database and graphics utility justify the Plus4's price tag.

The Plus4 and Commodore's other recent release, the C16, are the products of the same internal research and development project. Therefore it comes as no surprise that the two machines are for the most part identical. The only noteworthy difference is in the provision of semi-conductor memory within the two designs. The low-cost C16 has only 16K of user RAM together with a 32K ROM that contains the BASIC and operating system. The Plus4 on the other hand has a full 64K's worth of RAM, the same 32K ROM as the C16 to provide BASIC and a further 32K ROM that contains the four applications packages.

From the above, it will come as no surprise, that software written for the C16 will also function on the Plus4. Those who think that there will be an official upgrade to convert the humble C16 into a Plus4 are mistaken however. Although on the surface there would be little difficulty in designing such a piece of hardware, this scheme does not fit into Commodore's marketing plans.

A way with words

In view of the importance of the built-in firmware, this review will break with convention and discuss these packages before moving on to assess the hardware of the computer.

The concept of building applications software into a home computer was pioneered by Acorn who introduced the world to the sideways ROM. Whether or not Acorn predicted the way in which the concept would take off over the last few years is not known, but it is now possible to obtain a vast range of software in the form of ROM. The advantage of the Plus4's soft-

PLUS 4 REVIEW

ware is that the four packages are integrated. Thus data can be passed between the various areas of the software and then Commodore are able to squeeze quite a lot of power into 32K by virtue of the fact that the different packages are able to make use of common routines.

Some commentators have said that, in stressing the advantages of the applications software of the Plus4 in their advertising, Commodore are setting the machine up as a competitor to the QL which is also supplied with a similar suite of software. I do not share this view and see the Commodore's natural rivals as the top-of-the-range MSX machines (the Sony implementation features built-in firmware) – the QL at present, and possibly for some time to come, appealing to an enthusiast rather than mass market.

To enter the wordprocessor, it is only necessary to press one of the machines four function keys, F1. This software is a derivative of the Easy Script software available for the CBM64. The wordprocessor is curiously unnamed by Commodore who simply refer to it as the Plus4's wordprocessor; straightforward but hardly zappy! It is a good example of the packages which are now available for many home micros. It has a full range of commands that facilitate the entry and subsequent editing of text files. The provision of four cursor keys to the left of the Plus4's keyboard is a definite advantage and means that none of the wierd and wonderful key assignments of some home wordprocessors have to be remembered. Another useful feature is the status information displayed along the lower edge of the screen. The wordprocessor has two modes of operation, TEXT in which information is entered into the system and COMMAND which provides the means by which system commands are issued. The COMMAND mode is entered by pressing the 'Commodore' key at the right hand edge of the screen.

The wordprocessor also allows files to be merged, either to print out very long documents as a continuous file, the maximum for any one file is 99 lines, or, for example, to produce personalised letters by merging a form letter with a series of 'data files' containing a list of peoples' names and addresses.

Following the criteria by which wordprocessors for home micros are judged, the Plus4 software performs well. The two major drawbacks are firstly the fact that only 37 of the possible 77 characters per line are displayed on screen at any one time, the screen scrolling around in order to display the appropriate section at any one time, and that many of the printer formatting commands appear on screen as codes and have no effect on the way in which the text is displayed on the screen. Both these factors mean that it is not until a document is printed out that the presentation of the material can be assessed. These failings though are shared by the majority of micro-based wordprocessors and are solely due to shortcomings in the hardware of the computer. To provide the display circuitry necessary to display 80 column text and the character generators to display italic or bold text would take any micro out of the home category and place it firmly in the business sector of the market.

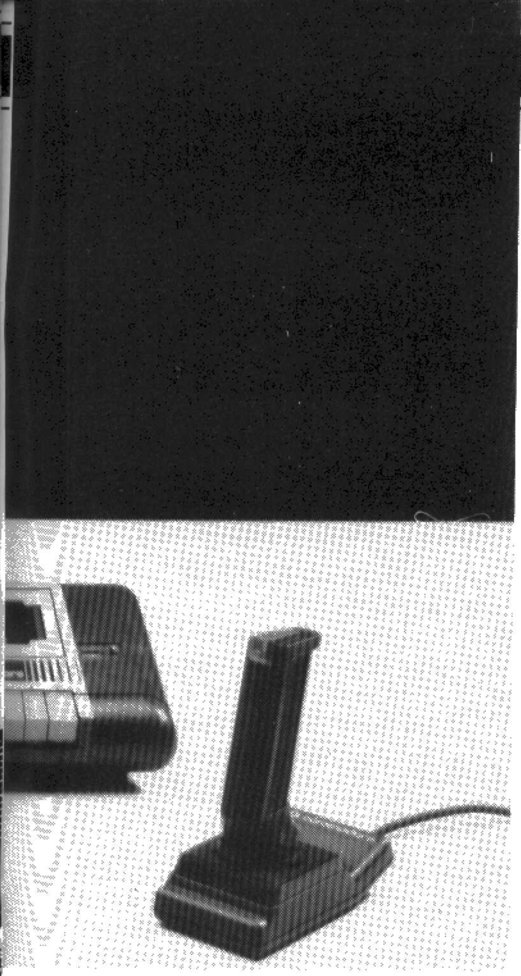
The Plus4's wordprocessor has a role beyond that of a text entry system in that it is the front end for the other 3 applications packages lurking within the computer. These are the spreadsheet, the data file manager and the graphics package. The spreadsheet is, in home computing terms, a very powerful piece of software. It is simply entered by issuing entering 'tc' while in the command mode of the wordprocessor. The maximum number of cells supported is 50 rows by 17 columns although only three rows of twelve cells are displayed on screen at any one time. Again, this is a limitation imposed by the hardware of the computer. Any cell may have up to 32 characters entered to it (only 11 of which are displayed on screen), the characters may be either text, numeric or to a formula that will produce output dependent on the content of other specified cells within the sheet. The spreadsheet has a very useful screen editor that greatly eases the task of getting data into the cells.

It is also possible for the spreadsheet to call upon the services of the graphics package by issuing the single command 'gr' from the word processors command mode. The graphics package is capable of producing both bar and dot graphs from the information within a specified series of cells.

The fact that the otherwise comprehensive manual devotes only five pages to the operation of the graphics package indicates that this is not the most powerful aspect of the Plus4's firmware (perhaps Plus3½ would have been a better name for the machine). The graphics generator will produce only the crudest of displays, the idea is that any output from this section of the software is passed to the wordprocessor where it is treated as a standard text file. All the niceties such as labelling are then added at this stage. It is a workable system and quite adequate for basic output but it is hardly the most elegant approach to the problems of providing graphic illustrations of the implications of the figures contained within the spreadsheet.

The last of the four packages will only be of interest to users who have purchased a disk system for use with their Plus4. While a disk drive will make the other applications software far more pleasant to use, it is possible to use them with a cassette recorder though. It cannot be argued that a disk file manager can be used without a disk however.

The disk file software is another example of a package which performs well in home computing terms. It allows the user to define the format in which the data is to be entered and to perform sorts on the data contained in any series of entries. It is also possible to pass the output of any sort to the wordprocessor for the purpose of producing well presented reports.



The four parts that go to make up the Plus4's firmware package can be said to be greater than the whole. The ability to freely pass data between various parts of the system with the minimum of fuss is a great bonus. That and the fact that the software is ROM based gives the Plus4 the edge over other micros that require tedious loading and system configuration before any work can commence. The question is just what demand will there be for the facilities provided by the firmware? Wordprocessing is an area that is growing in popularity and it is likely that many Plus4 owners, both in the home and business will find immediate use for this aspect of the Plus4. The other three sections of the system are unlikely to find much application in the home, the days when people use a spreadsheet to calculate the effect of mortgage increases on their financial position are still some way off as is the time when every housewife/husband keeps track of the stock in their larder with a data base. The majority of the power of the machine will thus only be appreciated by the small business user and they may find that while the Plus4 is a good home micro, they soon outgrow the power of the computer.

Hardware high points

The Commodore-owned IC manufacturing facility has come up with yet another derivative of the well known 6502 processor which has been designated 7501. This is driven by a variable speed clock that can run between 0.89 and 1.76MHz, the speed changing according to the type of memory (RAM or ROM) that the computer is addressing. The chip count of the Plus4 has been dramatically reduced from that of the CBM64, with the result that in addition to the processor, the eight dynamic RAM chips, four ROM ICs and the I/O devices, there is not a lot else worthy of comment.

To the outside world, the computer has a well engineered look to it. Commodore have long since learnt the importance of a typewriter-style keyboard that has a positive feel. A full length space bar is also regarded as an important feature. At the top of the keyboard there are four function keys which, when used in conjunction with the shift key, are capable of providing seven pre-programmed functions in addition to the help facility. It is good to see that the Plus4 provides both an on/off and a reset key on the right hand side of the machine. The majority of the I/O ports are brought out at the rear of the computer, these include the low voltage DC input from the separate power supply, a serial port, the cassette port, I/O port, memory expansion slot (into which additional ROM-based software may be installed) and the two joystick sockets plus the video output.

In common with both the VIC20 and CBM64, the Plus4 requires that a purpose-built, rather than standard audio, cassette recorder is used for program storage. As suggested above though, in order to get the most from this particular computer it requires that a disk drive be used rather than a data recorder. The well known 1541 disk drive can be used with the Plus4 and Commodore are to launch an updated version of this, the 1542 specifically for the new computer. In the new year there will also be a fast (by Commodore standards) disk drive designated the SFD-481; this, it is claimed, will speed up data transfer by a factor of twelve times.

In view of the fact that the Plus4 shares much of the same technology as the games orientated C16, it is no surprise that the computer offers an impressive performance as far as screen graphics and sound are concerned. The machine can reproduce eight luminance levels of 15 basic colours in addition to black, giving a specified total of 121 different colours although most people would have difficulty in telling the difference between some of the shades of colour. The one retrograde step is that the sprites of the 64 do not find a place on the Plus4 but to make up for this deficiency the problems associated with the creation of high resolution graphics on the company's current best selling computer have been addressed by the inclusion of a number of new BASIC commands. These include commands that will be familiar to owners of other computers and include DRAW, BOX, and CIRCLE.

The Plus4 brings an unfamiliar facility to the computer in the shape of a command that allows users to define on screen windows in order to display the output of different sections of programs on screen simultaneously.

Another welcome addition to the facilities offered by the Plus4 is the machine code monitor going under the name of Tedmon. This allows those familiar with the concepts of 6502 machine code programming to examine and shift blocks of code as well as being able to assemble, and disassemble single lines of code.

In view of the fact that in order to get the most from the applications software supplied with the computer it is essential to use a disk drive. It comes as no surprise that the new version of Basic provides a number of disk utilities that allow the user to get the most from the disk system.

The provisional documentation that we were supplied with for the purposes of review was divided into two distinct sections. The first of these dealt, in a fairly thorough fashion, with the initial setting up of the computer and with a step by step guide to the various commands of the Basic. The second half of the manual concerned itself with tutorials describing the operation of the built-in applications programs. Again this section of the manual was easy to follow and should mean that most users will soon be able to master the facilities offered by the packages.

In conclusion

Commodore have long since learnt the importance of packaging and the Plus4 continues in the traditions of the CBM64 in offering a well engineered product with good quality keyboard. The new version of Basic supplied with the machine is, on the whole, an improvement over its predecessor although the omission of sprites is a retrograde step.

The machine will stand or fall, though, on the acceptability of the built in firmware. This does not come cheap as price comparisons with the cost of the C16 seem to indicate that each of the four packages is costing the user around £40. While paying this sort of money for a wordprocessor will be justified in the case of many home users the other packages would seem to be of less immediate worth. Whether or not it would have been better for Commodore to restrict themselves to the provision of the wordprocessor and to have reduced the price of the machine by £100 or so is something that will cross many people's minds. The other items of software could have been supplied as plug-in cartridges for users to buy as and when they had a need for the additional facilities.

As yet there is little cassette-based software available for either the C16 or Plus4 although undoubtedly there are many software houses working on conversions of existing titles for these new machines. It is probable that both of the new computers will take some time to gain acceptance in the market and it will probably be next year before either product begins to see a high volume of sales. In the meantime though Commodore have their 64 machine which looks set to become, on a world wide basis, one of the most popular micros to date.

QL ASSEMBLERS

Adam Denning compares two QL assemblers and discovers that a prestige name is no guarantee of a prestige product.

Software development on any new machine is going to be very slow until a reasonable assembler is widely available. Last month Metacomco released a QL assembler and editor and very soon Sinclair is due to release an assembler written by CET, the company which wrote the alternative operating system for the QL known as 68K/OS.

We managed to get hold of both these products and gave them a thorough test. The results are perhaps a little surprising as the product with the prestige name (ie Sinclair's) is not as useful as the Metacomco product.

Metacomco's assembler is written in BCPL and was moved across from a larger 68000 machine without much effort. It comes on microdrive cartridge with two manuals – one for the screen editor and one for the assembler. Both products are capable of multi-tasking, although the length of the assembler means that it is very unlikely that anything other than very minor jobs, such as a clock, can run at the same time until expansion RAM is available.

"... the GST assembler makes one fatal error".

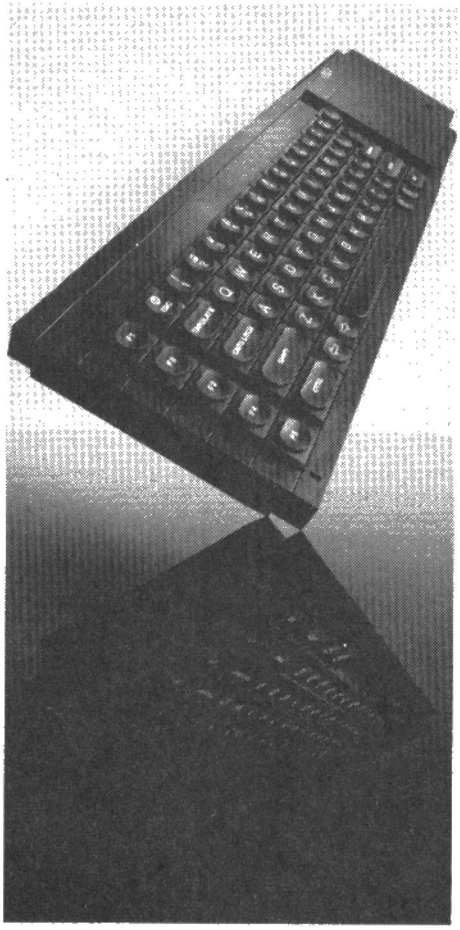
The editor is run by typing EXEC mdvn_ ed and is a derivation of Dr Tim King's renowned screen editor that has been available on the BBC Micro for some time with the BCPL development system available from Acornsoft. It is a very versatile product that utilises the QL's potential to the full. In common with the assembler the most noticeable thing about the product is that it is pleasant to use.

Invocation of the editor results in prompts for the name of the file being edited, so that already existing files can be altered or new files created, and then you are asked for the size of the editor's workspace. This can be defaulted by pressing ENTER, but larger files will need correspondingly larger amounts of workspace. The editor will accept decimal numbers like 20000 or abbreviations like 20K.

You are then given the option to alter the window size, and this is done by moving the window around with cursor keys, changing its shape where appropriate.

Once the window has been set the file is loaded if it exists, otherwise it is created. There is then a whole host of available commands, some of which can be entered and executed directly, others of which need 'command mode' to be entered first. All the usual screen editing functions are here, including global and selective search and replace, delete block, move block, save block and file merge. Tab settings can be altered and the cursor can be moved to any part of the text with a speed that should prove the embarrassment of Psion for evermore.

Alteration of files that already exist does not mean that you have to delete the old file before resaving, as you would in Basic, as the editor does it all for you. There is even an option to restart the editor with a different source file.



This editor is the ideal tool with which to create source files for the assembler and indeed it is suitable for almost any document or source creation, whether it be a manual, SuperBasic programs or informal letters.

The assembler follows the same format, using prompts in all the likely places, and listings can be directed to the screen or any other output device. The code can optionally be generated, and if it is then you can specify where it goes (from a file point of view rather than a RAM point of view, as no assembler of this scale performs assembly to RAM). The code produced is optionally absolute, position independent or relocatable. For almost all applications the second option is more preferable.

The assembler uses standard Motorola mnemonics and follows the usual labelling conventions except that periods (full stops) are part of the assembler syntax and therefore illegal in labels. This is occassionally frustrating, but only because the QDOS documentation uses periods in its labels.

Macros and libraries are also supported, and the whole comes across as a very professional product. It costs £59.95 and can be obtained from Metacomco, 26 Portland Square, Bristol.

The GST assembler makes one fatal mistake – it saves on RAM by leaving out facilities. A 68000 assembler necessarily has a large number of error messages – this one has over 50. While the Metacomco assembler reports its errors in full, with useful messages, this one just prints out a hex number. You must look up the meaning of this error in the manual.

"... as always, you get what you pay for ..."

The GST assembler is only 17K long, whilst Metacomco's is around 60K, but there is very little need to save on RAM when the host machine comes with 128K built in. Naturally the GST product assembles as well as the Metacomco one, but it just isn't friendly. Rather than being prompted for file name and listing devices you must actually specify the whole lot in the command line that you are asked for when the assembler is invoked. Changing the window size is just as troublesome, as you can't alter it dynamically. It expects the window size to be specified as a console specification (as in CON_400x200a36x10).

The GST assembler costs £39.95 and is available from the usual Sinclair suppliers.

In conclusion the more expensive product is very much the better, and the author's own programs are written and assembled using it. The article on extending SuperBasic with machine code elsewhere in this issue is as good a vindication of Metacomco's assembler and editor as any other.

HUNDREDS OF NEW LINES
MAPLIN
1985 Buyer's Guide to Electronic Components
PRICES ON THE PAGE Price £1·35
NOW WITH PRICES ON THE PAGE including Heathkit section
PRICE £1·35
BUYER'S GUIDE TO ELECTRONICS AND HOME COMPUTERS
£1·25

The amazing Maplin Catalogue is here again! The new edition is packed with hundreds and hundreds of new electronic components to bring you right up to date with all the latest developments. As all home constructors agree (and a good many professionals too) the Maplin Catalogue is the one essential piece of equipment they really need. And now with all our **prices on the page** the Maplin Catalogue is better value than ever.

On Sale From 10th November 1984.

Pick up a copy as soon as it's published at any branch of W.H. Smith or in one of our shops. The price is still just £1.35, or £1.75 by post from our Rayleigh address (quote CA02C).

Post this coupon now for your copy of the 1985 catalogue.
Price £1.35 + 40p post and packing. If you live outside the U.K. send £2.40 or 11 International Reply Coupons.
I enclose £1.75.

Name ....

Address ....

....

ECM/11/84
....

MAPLIN
ELECTRONIC SUPPLIES LTD

Maplin Electronic Supplies Ltd. Mail Order: P.O. Box 3, Rayleigh, Essex SS6 8LR.
Tel: Southend (0702) 552911. •Shops at: 159-161 King Street, Hammersmith,
London W6. Tel: 01-748 0926. •8 Oxford Road, Manchester. Tel: 061-236 0281.
•Lynton Square, Perry Barr, Birmingham. Tel: 021-356 7292.
•282-284 London Road, Westcliff-on-Sea, Essex. Tel: 0702-554000.
•46-48 Bevois Valley Road, Southampton. Tel: 0703-25831. All shops closed all day Monday.